AD A069210

① 6

EVALUATION OF AN EXISTING COMPUTER SYSTEM USING STRUCTURED ANALYSIS TECHNIQUES

DDC
RECEIVED
JUN 1 1979
A

THESIS

AFIT/GCS/EE/79-3   Dennis L. Schweitzer
Captain   USAF

79 05 29 127

6 EVALUATION OF AN EXISTING COMPUTER
SYSTEM USING STRUCTURED
ANALYSIS TECHNIQUES.

9 Master's THESIS,

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Wright-Patterson Air Force Base,

Ohio

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

11 Mar 79

12 161 p.

by

10 Dennis L. Schweitzer B.S.

Captain   USAF

Graduate Computer Science

Approved for public release; distribution unlimited.

012225

Preface

Software engineering techniques are being increasingly utilized to produce low cost, reliable computer systems. These methods offer a structured approach to define the system requirements and the system design.

I have attempted in this thesis to employ some of these techniques in a different application from that in which they are normally used. Rather than using them to define a nonexisting system, I began with an existing computer system, and attempted to evaluate its success at meeting user expectations.

I have assumed that the readers of this thesis are familiar with computer terminology. A basic knowledge of software engineering principles is also assumed.

For their assistance and for allowing generous usage of their equipment, I would like to express my gratitude to the Air Force Flight Dynamics Laboratory, and my sponsor, Mr. Bernie Groomes. For his guidance and encouragement, I would like to thank my thesis advisor, Maj. Alan Ross. For supplying great moral support, I offer a special thanks to my lovely wife, Suzi.

DENNIS L. SCHWEITZER

Contents

List of Figures

List of Tables

Abstract

This thesis is an evaluation of an existing computer graphics system using structured analysis techniques. The system evaluated is the IMLAC PDS-4 minicomputer at the Air Force Flight Dynamics Laboratory, Wright Patterson AFB.

The technique used in the evaluation is performed in three steps. First, the existing system is modeled into its logical equivalent using bubble charts. This model shows the flow of data through the system, and the activities performed on that data. Second, a similar model is produced describing the required system as defined by the users. This model is a representation of the users' desires and requirements, and is similar to a model which might be produced during the requirements definition phase of a computer's life cycle. The third step is to compare the two models to evaluate the existing system's effectiveness, and reveal areas requiring modification for full utilization of the system.

Following the evaluation, several recommendations and changes are proposed to increase the system's capability to meet the user requirements.

EVALUATION OF AN EXISTING COMPUTER
SYSTEM USING STRUCTURED
ANALYSIS TECHNIQUES

I. Introduction

Objectives

Structured analysis is recognized as a valuable tool
for performing the requirements definition phase of a system's
life cycle. When used properly, this technique provides a
step-by-step procedure for completely specifying the system
parameters. These parameters will reflect the four basic
principles of software engineering: modifiability, efficiency,
reliability, and understandability (Ref 8:21).

The purpose of this thesis is to employ the approach
of structured analysis towards a different application. A
formal set of procedures is developed to apply structured
analysis principles to the evaluation of an existing computer
system. The goals of this evaluation are to determine specific
areas which do not satisfy the requirements established by
the users of the system, and to define what modifications
are necessary to meet these requirements. The system to be
analyzed is the IMLAC PDS-4 graphics system located at the
Air Force Flight Dynamics Laboratory, WPAFB, Ohio.

Scope

The emphasis of this thesis is on the development and

1

examination of a formal methodology to perform evaluations
of existing computer systems. The IMLAC system is used to
illustrate an application of this technique, and to measure
the effectiveness and feasibility of employing the approach
to a "real world" problem. The results of the IMLAC evalu-
ation are presented to the users of the system as a set of
recommendations and modifications.

Approach

Chapter II contains background information about sev-
eral different structured analysis techniques available, and
outlines the proposed structured approach for evaluating
existing systems. Chapter III describes, in detail, the
different steps of the analysis technique, and applies these
to the IMLAC system. A brief history of the IMLAC PDS-4
system is provided as background of the system to be evalu-
ated. Chapter IV consists of a summary of the results ob-
tained from this study.

II. Background

Structured Analysis

Accurately def'ned system requirements are imperative
to a successful system development. The process of esta-
blishing these specifications is termed the requirements
definition phase of the development cycle, and encompasses
all aspects of the developmental process prior to the actual
design of the system (Ref 9:6). Failure to adequately specify
the requirements can result in increasingly expensive correc-
tions at later stages of development (Ref 2:5-6).

Currently, requirements are generated in free-form
English using a combination of ad hoc manual analysis and
common sense (Ref 2:5). Several problems that arise using
this method are (Ref 4:10-13):

1. Communication. The relationship between a user
   and analyst is complicated by the lack of common
   language, and by the inherent difficulty in des-
   cribing a non-physical object.

2. Changing requirements. The user requirements
   change frequently as the user becomes more aware
   of what capabilities are available.

3. Lack of tools. The only tools available to the
   analyst are paper, pencil, and his mental ability.

4. Problems of documentation. Managing the enormous
   amount of text required to describe a complex
   system is a major problem in requirements analysis.

5. Work allocation. A large complex problem does not
   normally divide into equivalent work efforts.

6. Politics. The introduction of a new system is
   often associated with management prejudices and
   power struggles.

Several techniques have been developed to deal with these problems. One major area of study is in the use of automated tools to specify and analyze requirements. The ISDOS system is one such tool which incorporates a problem statement language (PSL) to reduce terminology inconsistencies, and a problem statement analyzer (PSA) to cross check for redundancies in specification (Ref 11). The analyst defines the system using the PSL which invokes strict interpretation of terms and relationships of objects. Several advantages are offered by this method. Interim reports and summaries can be automatically produced, organization of information is automated, and consistency and completeness are guaranteed. The development of an extended version of ISDOS is being sponsored by the Air Force, and is known as CARA (Ref 12). A similar automated approach, known as SREP, is being used by the US Army Ballistic Missile Defense Advanced Technological Center (BMDATC) for real-time developments (Ref 1).

Another area of study is the use of metalanguages to state specifications (Ref 13:214). Restrictions placed on these languages promote comprehensibility and correctness of proof. Examples of such languages are EUCLYD, CLU, ALPHARD, and the higher order software (HOS) axioms.

SofTech has developed a rigorous manual method for requirement specification known as structured analysis and design technique (SADT) (Ref 1∅). The central concept of SADT is to decompose a problem into a series of conceptual models. These models are generated utilizing the basic building blocks of all SADT diagrams, boxes and arrows.
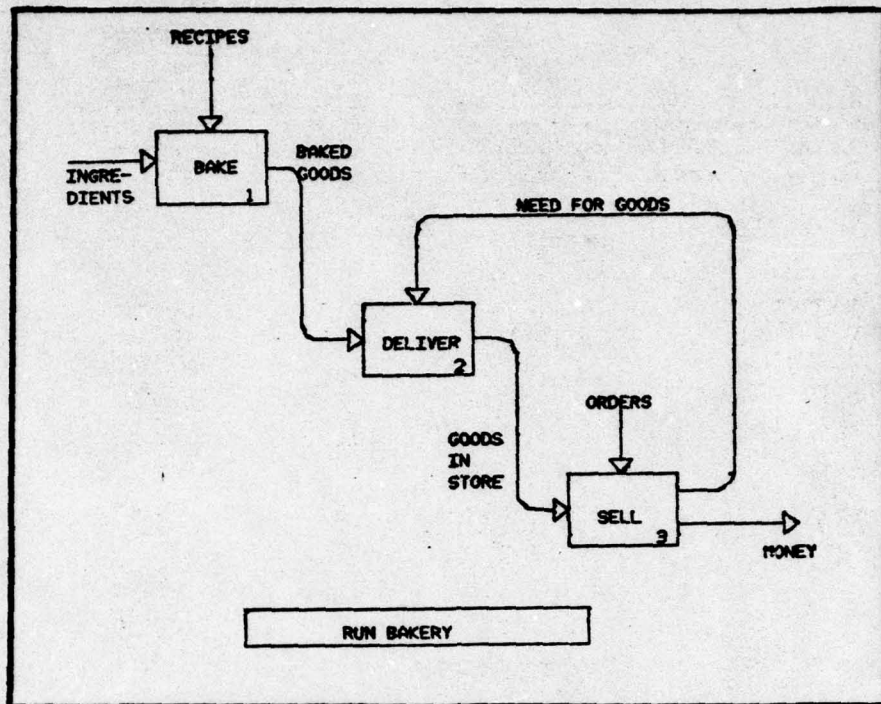
Figure 1. SADT Example

Boxes represent processes performed on data, and arrows represent the data. A simple example is shown in Figure 1. For an entire system, each process must be represented by a box. To facilitate the large number of such processes, and to promote readability, a top-down structured decomposition is used for the entire system. That is, the entire system is initially represented as one activity box. A second level will decompose this activity into 5-7 sub-activities. Each of these will be further decomposed, and so forth, until the entire system is modeled to the desired level of detail. Figure 2 shows this decomposition graphically (Ref 1∅:2-3).

Advantages of this method are its readability, modularity, and structure. It provides a straight forward
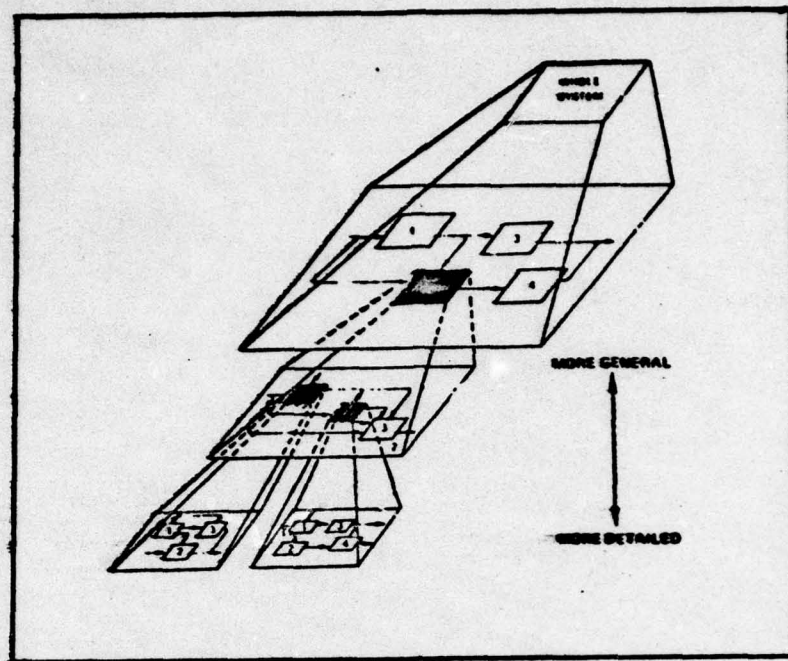
Figure 2. Structured Decomposition

step-by-step procedure for breaking a problem into a series
of sub-parts with well defined interconnections.

De Marco has developed a similar manual method known
as structured analysis (Ref 4). This technique also uses
hierarchal models called "data flow diagrams" (DFD's). These
are identical to the SADT models in concept, but with dif-
ferent building blocks. Instead of boxes to represent pro-
cesses, structured analysis uses bubbles. The elements are
shown graphically in Table I. The model similarity is such
that De Marco describes SADT as "...data flow diagrams with
square bubbles." (Ref 4:48). The final output of structured
analysis is a "target document" consisting of DFD's, a data
dictionary defining file compositions, and structured English.

Structured analysis is normally employed as a means to

6

TABLE I

Structured Analysis Building Blocks

Symbol	Name	Description
◯	Bubble	Process performed on data
⌒→	Data Flow	Data item
—	File	Data storage
▢	Source/Sink	Place where data originates or terminates
✳/⊕	And/Or	Relational operators on data

specify requirements for a new automated system. The analysis is divided into seven component studies (Ref 4:27):

1. Study the current physical environment

2. Derivation of a logical equivalent of the current environment

3. Derivation of a new logical environment

4. Determination of physical characteristics to produce a new set of physical environments

5. Quantification of cost and schedule factors

6. Selection of one new physical environment

7. Packaging the structured specification.

Figure 3 shows these components in a data flow diagram (Ref 4:26). Each step will be briefly discussed.

The current physical environment that is being studied consists of the non-automated process. The analyst must

7

Figure 3. Structured Analysis Flow

become familiar with the flow of information and output
products in order to understand the nature of the system.
The product of this study is a DFD which models the physical
system as viewed by the user.

The next step is to derive a logical equivalent of the
physical model. This is necessary to define the logical
functions which are to be performed by the system. User
interaction is important here to assure an accurate repre-
sentation of the functions performed. A logical DFD is the
output of this effort.

8

Derivation of the new environment is the user's op-
portunity to specify what additional tasks or modifications
are desired of the new system. A data flow diagram is pro-
duced which totally defines the logical functions the system
is to accomplish.

At step four, physical characteristics and boundaries
are introduced to design several alternative physical models
of the new system. These characteristics are specified by
the user as required physical limitations. As stated, sev-
eral alternative models are produced to illustrate physical
trade-offs.

Selection of one model from the physical set requires
several studies of costs and schedules. Each alternative
must be quantified based on the goals and constraints of the
organization.

The final component of the analysis is the packaging
of the documentation and specifications. This package is
necessary for the system design phase, and represents the
main product of the analysis portion of the software life
cycle.

Structured Analysis as an Evaluation Tool

Structured analysis provides several advantages for
defining requirements of a new system. The models provide
a top-down logical overview of the functions performed.
Modeling the current physical system ensures that all neces-
sary functions are considered, and the seven steps provide
a structured sequence of procedures to define the require-
ments.

Several organizations possess existing systems which may or may not be fulfilling the requirements of the users. Failure to meet these needs may be a result of changing requirements or incomplete initial specifications. In either case, some method is needed to evaluate the existing system against user requirements. Several advantages afforded by structured analysis can be applied to this type of evaluation. Graphic models provide an easy-to-read concise representation of a system. Modeling the system in a logical manner encourages a functional evaluation rather than a physical one. Also, a step-by-step methodology promotes completeness and accuracy.

The structured analysis procedures listed earlier cannot be applied directly to an evaluation effort. This is a result of the fact that the existing system is already automated, and is not to be replaced by a completely new system. However, several of the same techniques and general goals can be used in a different sequence. The following procedures are presented as a possible application of structured analysis techniques to an evaluation:

1. Logical modeling of the current system.

2. Logical modeling of the required system.

3. Recommendations based on model comparison.

Each phase will be discussed briefly, and a detailed application will be presented in the next chapter.

Modeling of the current system is the equivalent of the first two phases of structured analysis. The purpose

is to become completely familiar with the existing system, and to model its logical equivalent in a data flow diagram. The creation of a physical DFD, as specified in structured analysis, is not recommended for an existing system. The level of detail involved in a complex system would result in an unmanageable physical model. It is more efficient to group the system details into logical equivalents first, and then model the logical system. The resulting DFD's provide a framework for comparing the existing logical system to a required logical system.

The required system model is a representation of the logical functions which the user specifies as necessary. It is recommended that this model be accomplished second so that the analyst is completely familiar with the existing system's capability and limitations prior to interpreting user requirements. The existing model is used as a guide-line for interviewing users as to what functions should be added or deleted. Modeling the required system in the same format as the existing system also promotes a straight forward comparison of the logical models.

Comparing the two models is a formalization of the discrepancies between the existing functions and the required functions. In this comparison, the analyst must decide if additional functions specified in the required model are valid and relevant. The result of this comparison is a set of recommendations from the analyst to the user.

For simplicity, the term "structured evaluation" will

11

be used to signify the methodology described. Chapter III presents an in-depth example of applying structured evaluation to an actual system. The separate phases will be discussed in greater detail, and the associated problems will be identified.

III. An Application of Structured Evaluation

IMLAC System

One mission of the Analysis and Optimization Branch (FBR) of the Air Force Flight Dynamics Laboratory is to perform and support structural analysis. To accomplish this mission, a general purpose graphics computer was requested in March 1977 (Ref 5). As a result of this request, an IMLAC PDS-4 computer and supporting equipment was leased in April 1977.

The IMLAC system is shown graphically in Figure 4 (Ref 7:4). It includes 32 K of 16-bit memory, a refresh vector graphics CRT, light pen, dual hard disk storage, and a Versatec printer. The system is capable of operating stand-alone for local processing, or as a terminal to a host computer with a standard RS-232 interface. Supporting software consists of several utilities and a Fortran compiler for stand-alone mode, and interfacing programs when used as a terminal. A more detailed description of the operating system is presented as a user's guide in Appendix D. This guide is a direct product of the IMLAC evaluation.

Justification for the IMLAC system by FBR was based on the specifications shown in Table II (Ref 7:5). The flicker-free performance capability, available Fortran, and disk capacity were reasons cited for sole source justification of the IMLAC system (Ref 6).

Since the installation of the equipment, several problems have surfaced resulting in limited use of the system.

13

Interactive device options

Light pen	Joystick
Trackball	Graphic Mouse
Data tablet	Function keyboards
Additional or special keyboards	

KEYBOARD

PROGRAMMER/
MAINTENANCE
CONSOLE
(optional)

RS232

ASYNCH

INT.

MAIN PROCESSOR

MEMORY
990 nsec/cycle
4 - 64K
16 bit word

DISPLAY PROCESSOR

MONITOR
17 INCH

Interface options

Synchronous
Multiple asynch.
Programmable speed
 select
Acoustic coupler
High speed serial
Three-speed switch

Parallel computer-interface options

Univac AN/UYK-7
Univac 1230
Harris 6024/5
DEC PDP-11
DG NOVA
DG ECLIPSE
HP 2100
Honeywell 516/716

MP options

ROM
ROM switch
Floating point
 module

Peripheral options

Cassette storage
Paper tape reader
Paper tape punch
Magnetic tape systems
Disk systems
Floppy disk systems
Disk share
Hardcopy
Printers
Plotters

Memory options

64K
Direct memory
 access

DP options

Circle/arc
 generator
Scale, Rotate,
 Trans., Window
Monitor control
 interface
Color monitor
 interface

Monitor options
----
15 or 21" monitor
Slave monitors
Independent monitors
Manual image control
Phosphors
Color monitor

Figure 4. IMLAC System

14

TABLE II

IMLAC PDS-4 Specifications

CRT DISPLAY

Screen Size	17" diagonal (21" optional)
Maximum Viewing Area	10" x 11" (13" x 14", 21" CRT)
Quality Viewing Area	9.5" x 9.5" (12" x 12", 21" CRT)
Brightness	50 foot lamberts
Intensity Levels	16 (black to maximum)
Contrast Ratio	20 to 1
Refresh Rate	40 frames per second. Refresh rate is program controllable. (30, 60 fps optional.)
Displayable Characters	3,000 (4,000 @ 30 fps)
Phosphor	P39 (P12 optional for 30 fps refresh rate)
Deflection Type	Magnetic
Spot Size	12 Mils, nominal
Repositioning Time	20 μsec full screen, nominal

DISPLAY PROCESSOR

Addressable Locations (Relative)	14,366 x 14,366
Maximum Display Resolution	2048 x 2048

VECTOR GENERATOR

Display (1024 resolution)	68,000 inches/second of vector, flicker free
Scissoring	Hardware
Vector Modes	Dot, Dash, Solid, Dot-Dash, Invisible
Hardware Blink	Under Program Control

Relative Vectors can be drawn at any angle anywhere

Specifications Subject to Change without notice

CHARACTER/SYMBOL GENERATOR

Stroke Drawn Characters	
Character Set/Symbols	Full 96 ASCII Set (upper and lower case) plus user-defined/programmed special symbols
Character/Symbol Rotation	0°, 90°, 180°, 270°
Writing Rate	Less than 10 microseconds, average

MINI-COMPUTER

Word Length	16 bits
Memory Size	4096 words, expandable in 4096 or 8192 blocks for 65,536 words
Cycle/Execute Time	990 nanoseconds
I/O	Serial asynchronous from 110 to 600,000 baud (RS 232 standard). Other serial and parallel interfaces optional, including MIL-STD 188C.

SOFTWARE

IMLAC also offers a complete series of Standard Utility and Support Software.

MECHANICAL/ELECTRICAL

Size	Self-contained desk-like console 36" W x 30" D x 28" H (48" wide tabletop optional)
Weight	150 pounds (approximate)
Power	117 VAC, 3-wire, single phase 60 Hz standard. Other voltages and 50 Hz optional.

Prospective users described several problems including:

1. Inadequate documentation

2. Non-standard Fortran

3. Hard-to-learn procedures

4. Confusing keyboard strokes

5. Incomplete graphics capability.

These problem areas, combined with others, dramatically reduced the utilization of the system. At the beginning of this study, just one person was using the stand-alone capability, and only in a limited sense. A few others were occasionally exercising the TTY capabilities.

Modeling Conventions

De Marco's modeling technique is used for representing the systems in structured evaluation. A brief discussion of data flow diagram conventions is presented here for clarification.

The basic building blocks of DFD's are shown in Table I (page 7). Bubbles are processes or functions that are performed on data. Data flows, symbolized by arrows, represent a path along which information passes. The data always flows in the direction of the arrow. Sources and sinks are originators and consumers of data. A file is a storage device for data. Relational operators are used when multiple data flows enter or leave a bubble. The operator indicates whether both data flows are necessary (AND) or only one (EXCLUSIVE OR).

Leveled data flow diagrams are a hierarchy of DFD's

16
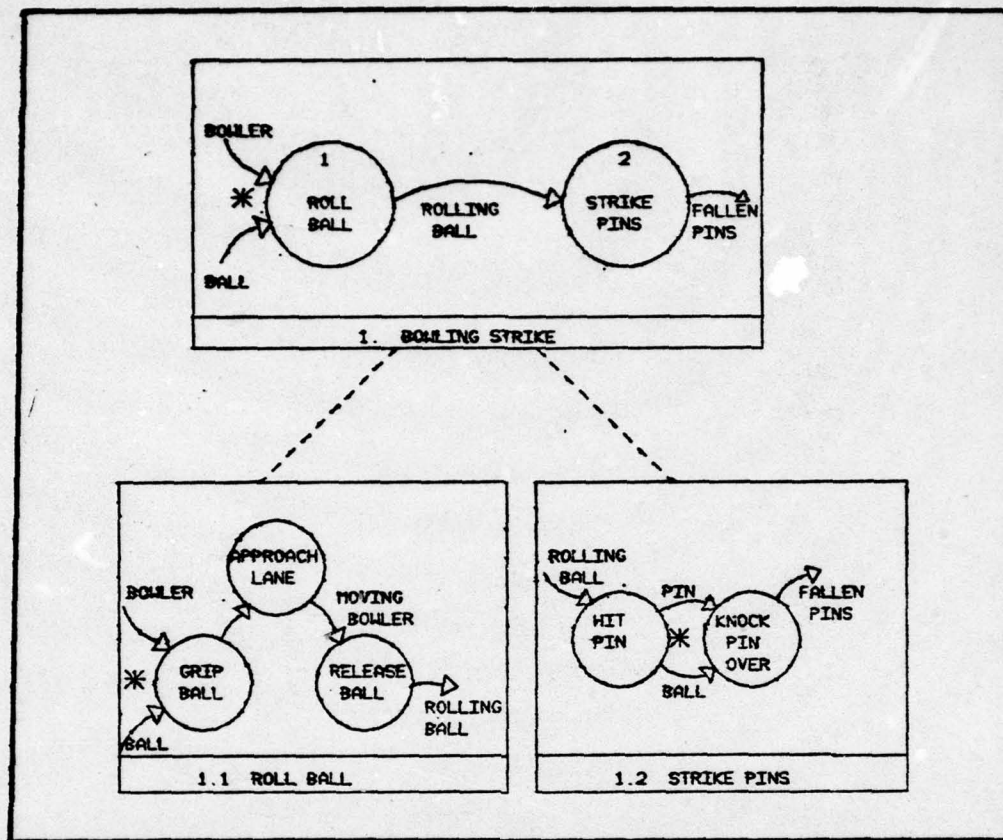
Figure 5. Example Parent/Child Relationship

with each bubble on the first level DFD being decomposed by a succeeding diagram. This hierarchy provides a top-down view of the system with each separate diagram encompassing a complete function. To reflect the relationship of the diagrams, a specific numbering scheme is used. An example is illustrated in Figure 5. Each high level DFD is associated with a number unique from other high level DFD's. Each bubble on the high level diagram contains a number unique from the other bubbles on the diagram. Second level DFD's are numbered with two numbers in the form X.X. The first number indicates which high level diagram is being decomposed in this

17

DFD, and the second number specifies which unique bubble on
the higher DFD is being expanded in the lower diagram. The
terms "parent" and "child" are used to describe this relation-
ship.

Existing System Model

The modeling of the IMLAC system was accomplished in
four steps:

1. Familiarization with the system

2. Identification of logical functions

3. Development of DFD's

4. Verification of model correctness.

Each of these will be discussed in detail.

System Familiarization. To accurately model any com-
plex system, it is essential for the modeler to become inti-
mately familiar with the details of the system. When the
system is a computer, this involves learning the functions
accomplished by the operating system, and the algorithms
used in performing this. One means of obtaining this know-
ledge is to write several assembly language programs, and
perform modifications to the operating system. This tech-
nique was used with the IMLAC. It required several months
to complete this phase of the modeling. To make the effort
more productive, the programs and modifications attempted
were suggested by the users as convenient additions. Thus,
information about the system was being acquired while providing
improvements to the system.

The assembly language programs were written with an emphasis on utilizing several available software utilities, and interfacing with all peripherals. For example, several utilities were written which used routines to write to the display screen, disk, printer, and TTY port. These assembly language programs enabled the analyst to better understand the operating system source code, and gain insight into the theory of operation of the system. Modifications to the operating system provided several insights into the algorithms used to perform system functions. Two such modifications completed include variable size character selection in the MONITOR, and improved print quality in local graphics. Several other modifications were investigated which involved in-depth source code study and a complete review of available documentation.

Identification of Logical Functions. To model the logical equivalence of a physical system, it is necessary to identify what logical functions are performed. A systematic approach was used to determine these for the IMLAC.

Initially, the names of all operating system modules were listed in a single list. Modules which accomplished similar tasks were then grouped together with a description of the common task. The criteria which were used to discern similar tasks included common peripheral utilization, and analogous duties performed. For example, KP, PK, MN, NM, and RENAME are all modules which modify files, and were grouped under the heading of File Modification. Next, all

19

modules left ungrouped were considered for inclusion in an existing group by slightly modifying the group description. For example, DELETE was added to the File Modification group by renaming it File Maintenance. Modules which were still ungrouped were defined as separate categories.

The next step in the function identification is to normalize the level of abstraction between the different groups. This is necessary to produce a list of functions which could be modeled at the same leveled DFD. To accomplish this normalization, an iterative procedure was used. First, categories which appeared as highly specialized were considered for regrouping with other categories or parts of other categories. If this could not be accomplished, the group was disbanded, and the modules were distributed to other categories based on their functions. Categories which appeared to be too general were considered for partitioning into separate categories. This process of regrouping and partitioning was continued until all categories appeared to consist of a similar level of abstraction. The final list is shown in Table III. These categories are the logical functions to be modeled in the data flow diagrams.

Development of the DFD's. De Marco offers the following guidelines for developing DFD's (Ref 4:63):

1. Identify net input, output, and data flows

2. Work outputs to inputs, or from the middle out

3. Label all data flows

4. Label all bubbles

20

TABLE III

IMLAC Logical Functions

Model Reference	Function	Modules
1.	Operating System	MONITOR, SYSLOD, ROMLOD, MAKELG, LOGIO, DIKY, DEBUG, START
2.	File Maintenance	DELETE, FLUSH, FILCOM, GET, MN, NM, PK, KP, DIREKT, DPRINT, PRINT, RENAME
3.	Local Graphics	TIS4, SYMBL4
4.	Host Graphics	TIS4, GTS, MMS
5.	TTY Mode	LINDI, PSA, STR14
6.	Disk Maintenance	COPY, SAVE, CHECK, SYSTID, STATUS
7.	Program Creation	EDITOR, ASSEMBLER, COMPILER, BINDER

5. Ignore initialization and termination

6. Omit trivial error paths

7. Do not show flow control

8. Be prepared to start over.

An informal version of these rules was used in drawing the DFD's for each identified logical function.

First, the net input and output data items were listed for the logical function being diagramed. Next, all duties performed by the modules were listed under each logical

heading. These duties were combined and generalized to produce a summarized list of processes performed on the data. Each of these processes was represented by a bubble with appropriate data flows drawn between them. It is important to note that the separate processes within each function do not necessarily correspond to one particular program. These processes represent a logical overview of the function, and each process may generalize several programs or parts of programs.

To complete the hierarchal structure of the model, each process, or bubble, had to be further decomposed into a second level of detail. This was accomplished in a similar manner as the high level DFD. A bubble was chosen for decomposition, net inputs and outputs were defined, and a list of functions was generated. It was decided to only model two levels of detail for this effort. This decision was based on the time restrictions for completing the study, and the belief that two levels would provide sufficient detail to compare the models. The complete model for the IMLAC system is given in Appendix A.

Verification of Correctness. It is necessary to perform several tests of verification on a completed model to ensure the accuracy of the representation. De Marco suggests the following five checks (Ref 4:1Ø6-112):

1. Check that all items are labeled
2. Check for consistency of abstraction level
3. Verify data conservation

22

4. Identify any file problems

5. Perform a model walkthrough.

The first check is a bookkeeping task to ensure model completeness. All data flows and bubbles were checked for proper labeling, and necessary corrections were made.

Abstraction consistency is a check that all models on the same level contain the same amount of detail. This consistency aids in readability and coherency of the model. The check was accomplished by verifying that all data flows and processes represented similar levels of abstraction.

The term "data conservation" refers to the requirement that all data flowing into and out of a second level DFD is shown on the DFD from which it is decomposed, its parent. In other words, only data flows associated with a first level bubble can appear as inputs or outputs to its second level child. The only exception to this rule is when the data is completely internal to the process being modeled. Conservation was completed by comparing parent and child DFD's, and exceptions were noted in the respective text.

File problems are characterized by such discrepancies as data only flowing into a file, or inconsistent data flowing into and out of a file. For the IMLAC, these problems required renaming some data flows.

A model walkthrough is the most time-consuming of the verification checks. Each model was reviewed by following the data from process to process, and ensuring that the diagramed flow was valid. This required checking the

23

documentation and source code of the modules involved against the DFD.

Summary. The structured analysis model of the existing system represents a logical top-down view of the physical IMLAC graphics system. Several problems were associated with the modeling process.

Familiarization with the system was the most time-consuming phase with no definite completion point. There is no measurable criteria available to indicate when the modeler knows the system well enough to model it. There was also a delay in this part of the effort while the source code was being ordered and received from IMLAC.

Identification of the logical functions also lacks measurable criteria defining a "good" portrayal of the physical system. Decisions were made on a subjective basis with no cross check available.

The development of the DFD's required many decisions about, content, meaningful data and process names, and adequate abstraction levels. Due to the logical nature of the model, there was not a one-for-one correlation of the DFD bubbles to the IMLAC software modules. This resulted in generalizations of similar processes performed by several different modules, and interpretations of the total function achieved.

The verification checks were effective in isolating several areas requiring further refinement. However, as in the other phases, numerous subjective decisions were necessary

24

to complete these checks. Since there were no personnel at the laboratory familiar with the system details, the final model could not be reviewed by a competent critic. Thus, the final model contains several personal viewpoints of the system as interpreted by the modeler.

Required System Model

To evaluate the system's performance, it is necessary to have some criteria against which to measure the existing system. This means there must be some formal description of the system parameters, as defined by the users. When a non-existing system is being developed, such an assessment of the needs to be fulfilled is known as the requirements definition (Ref 9:6).

To provide a common ground for comparison, a model of the requirements is built similar to the existing model. The same techniques used for the existing system model are repeated, but with some variations which will be described.

System Familiarization. Since the required system does not exist in a physical sense, there is no code that can be studied, nor algorithms to learn that reveal the system functions. These properties can only be discovered by interviews with the eligible users of the system. Some information was obtained from the initial request for the system (Ref 5), but the specifications were too general for a detailed model.

Similar interviews were performed with several users.

Initially, general questions were asked concerning how the system was to be used, and what types of capabilities were considered necessary to satisfy performance needs. Following these responses, more specific information was requested about the details of the desired system. Emphasis during the interviews was in terms of how a prospective system might operate, not in terms of the existing system.

Identification of Logical Functions. Data flow diagrams are a logical representation of the system. Thus, as in the existing system model, it is necessary to identify the logical functions required to fulfill the users' needs. However, since this model is to be compared to a different set of DFD's, it is desirable that some degree of similarity exist between the functional breakdown of the two systems. If all of the user requirements can be interpreted as belonging to one of the existing model's functions, it is advantageous to use the same set of logical functions. This set of functions provides a direct means of comparing the two models and an initial framework for modeling the required system based on the existing system. For this study, the existing functions were sufficient to incorporate the user requirements and were used for the DFD's. This sufficiency was determined from user interviews concerning the general goals and usages of the desired system.

Development of the DFD's. The procedures used for generating the diagrams followed the same sequence as for the existing system model.

26

First, net inputs, outputs, and processes were identi-
fied. This involved interpreting the user supplied infor-
mation in relation to the responsibilities normally accom-
plished by each logical function. Unless the user speci-
fically indicated some capability or function different from
the existing system, the appropriate DFD from the existing
system was assumed to be sufficient. Where differences
existed, the desired system was drawn using the procedures
outlined in the existing system section. In several in-
stances, this resulted in simply adding extra bubbles to
the existing diagram.

The completed model is shown in Appendix B. To avoid
repetition, those DFD's identical to the existing system
model were not redrawn. These are identified in the asso-
ciated text.

Verification of Correctness. The five verification
checks described earlier were also applied to the required
system model. The first four were completed in an identi-
cal manner as for the previous model. The walkthrough,
however, could not be accomplished the same way, since there
is no documentation or code to compare with the model.
Ideally, this would be done by having the specifying user
review the model. However, such a review would require the
user to become familiar with the modeling structure and
protocol. Within this study, the primary users were con-
strained in the time available to learn the modeling struc-
ture, so a simpler approach was taken. Each functional DFD

was reduced to a narrative description, and verbally confirmed with the user as accurate. Any discrepancies or changes were incorporated into the DFD.

Summary. The requirements definition model provides a concise description of a required system in a format compatable to the existing system model. The development of this model presented a different set of problems than those described for the existing system model.

Attempting to extract useful information from the user was perhaps the greatest challenge. It was difficult to get the user to talk in terms of functional capability rather than specific physical desires. Another problem encountered was similar to the "second-system effect" as described by Brooks (Ref 3:55-58). Users tended to view the existing system capabilities, and suggest several "frills" to add on.

Problems with the development of the DFD's were similar to those for the existing system. Several subjective decisions were necessary to complete the model.

The described walkthrough procedure resulted in some communication problems. The user was unable to understand all concepts explained in a DFD narrative description. Several specific ideas in the DFD had to be expanded before a questionable point could be resolved.

The completed model represents a system that would satisfy the user requirements. The next stage of the evaluation is to compare this model with the existing system.

Comparison of the Models

The purpose of structured evaluation is to provide the user of an existing system with a set of recommendations for improving the effectiveness of the system. These recommendations are based on a comparison of the existing model and the model of the required system.

Data flow diagrams are a functional decomposition of a system. Attempting to compare two such models is meaningless unless there exists some similarity in the functional breakdown of the two systems. Although the defined functions do not need to be identical, the required system should be either a subset or superset of the existing system. This provides for comparison of similar functions, deletion of existing functions which are not required, and addition of new functions not already existing. The only difference in the functional breakdown of the two IMLAC models is the lack of a Disk Maintenance function in the required system. This is a result of no user defined differences from the existing system. Procedures for completing the comparison will be briefly discussed.

The first step in comparing the two logical models is to select a function and review the first level DFD's for any differences. Since the first level DFD's represent a general overview of the function, differences at this level usually indicate a discrepancy in the concept of the function. That is, either the function is not complete, or the function contains procedures which may be contrary to the requirements

of the user. The Host Graphics function (Diagram 5 in Appendix B) of the required system is an example of this type of discrepancy. The function has been expanded in this model to include software routines on the host machine. All discrepancies in the first level decomposition should be marked for consideration in the second level review. Examples of such differences include changed data flows and additional outputs.

The second step in the model comparison is to sequentially select and review all second level decompositions of the chosen function. The corresponding DFD's should be checked for any discrepancies, and noted accordingly. It is important to consider differences imposed by changes in the high level parent. Differences at the second level indicate a discrepancy in the realization of a function rather than in the concept. For example, in Diagram 1.1, both models depict the concept of interpreting user commands. The required system delineates between regular commands and special purpose commands.

After all second level DFD's have been reviewed, the third step in the comparison process begins. All noted discrepancies for the selected function must be reviewed for their relevancy, and meaningful recommendations generated. An irrelevant discrepancy is one that does not affect the operation of the system. For example, the Operating System function (Diagram 1) shows a file and data flow discrepancy. However, this particular difference is only in the existing system implementation, and has no effect on the desired capabilities

of the required system. Thus, it is disregarded for recom-
mendation. Another type of irrelevant difference is an added
capability which in actuality exists in a different form. For
example, Diagram 2.4 shows an added capability of multiple
file deletions as a comparison difference. This capability
exists in the actual system, but failed to show up in the
model because it requires a combination of manual file-naming
conventions with the automated capability. Such irrelevant
discrepancies can be a result of deficiencies in the model, or
of a lack of understanding about the system capabilities by
the user. Several such differences are avoided when the re-
quired model is created by explaining to the user how different
features can be implemented on the existing system.

Deriving meaningful and useful recommendations from a
set of discrepancies is a difficult task. Several consider-
ations must be accounted for in suggesting improvements to the
existing system. Two such considerations are: the feasibility
of the recommendation, and the resources available to the user
to accomplish the recommendation. There is no set sequence
of procedures to develop recommendations from the comparison
discrepancies. To organize the recommendations, three cate-
gories are used: hardware, software, and procedural.

Hardware recommendations are based on expanded capa-
bilities to facilitate desired improvements. An example is
the addition of an in-office host minicomputer to resolve
host graphic requirements. Software recommendations include
specific programs to be developed and modifications to some

existing utilities. Procedural suggestions encompass documentation, system operating policies, and other aspects of management.

The complete set of discrepancies and associated recommendations for the IMLAC is presented in Appendix C. These lists are a direct result of structured evaluation, and are presented to the Evaluation and Optimization Branch of the Air Force Flight Dynamics Laboratory for consideration.

IV. Conclusions

Summary

Several software engineering techniques are available
to define system requirements in a life cycle. The advantages
achieved by these techniques are desirable in an evaluation
of an existing computer system. To realize these advantages,
a variation of De Marco's structured analysis methodology is
developed for application to an existing system. This derived
technique, known as structured evaluation, is applied to an
IMLAC PDS-4 graphics system for illustration. Structured
evaluation consists of three phases: current system model,
required system model, and model comparison.

The current system model is a graphical representation
of the functions performed by the existing system. The mod-
eling format is identical to the structured analysis technique,
and consists of a series of decompositions, called DFD's,
which depict the functional system in increasing detail.
Modeling is accomplished in four stages: system familiarization,
function identification, DFD development, and model verifi-
cation. The IMLAC model contains two levels of abstraction,
and is shown in Appendix A.

The required system model is similar to the current
model, but represents system requirements as defined by the
user. The same four modeling stages are repeated for model
creation, but with different methods of implementation.
Familiarization is performed through user interviews about

desired capabilities, mission requirements, and existing
problems. The same logical functions are used for both models
in this study to enhance comparison and provide a framework
for interpreting user requirements. The required IMLAC model
is also restricted to two levels of detail, and is shown in
Appendix B.

Model comparison is a procedure of defining discrep-
ancies between the existing and required models. This pro-
cedure is accomplished through a systematic review of the
corresponding function DFD's. Resulting conflicts must be
inspected to isolate deficiencies in the existing system.
Possible solutions to the discovered deficiencies are examined
in regard to available resources and feasibility. A set of
discrepancies and recommendations for the IMLAC is presented
in Appendix C.

Observations

Modeling the existing system in structured diagrams
proved an effective tool for learning the entire system. All
aspects of the operating system had to be thoroughly investi-
gated to complete the model. Any areas which were not as
well understood became immediately obvious when modeling that
particular aspect. Examining the system details as the model
progressed provided a sequenced approach to investigating the
entire system. This structure supplied a framework for ac-
complishing the enormous task of learning an entire operating
system.

Definition of logical functions forced the analyst to

34

view the existing physical system at an abstract level. This provided an understanding of the relationships between individual components and of the global structure of the system. An example of this viewpoint is illustrated in the developed user's guide in Appendix D. The system is viewed as a series of functional areas rather than individual programs. Generation of these functions was not a straight forward procedure, but required several redefinitions. Although a goal of the defined functions is to maintain a common level of abstraction, this could not be easily discerned until the actual modeling began and details were investigated. Thus, it is possible to discover after completing several DFD's, that the functional breakdown is not satisfactory, and the effort must be restarted.

Modeling the existing system with only one analyst created a problem in model accuracy. Several decisions were arbitrary in nature based on personal interpretation of the functions. Since no other person could verify these decisions, there was no cross check for accuracy. Having only one analyst resulted in another disadvantage concerning the value of the final model. The model is a concise representation of the system and could be a useful tool for documenting the flow of the system, but only to someone familiar with the modeling technique. Since the users of the system have no training in understanding the model, several possible advantages are lost.

Modeling the required system was simplified by using

the same functional descriptions for the system. Interviews were conducted using the existing model as an initial guide for questions. Development of the DFD's started with the existing diagrams as an initial position and modified them accordingly. In several instances, the existing system DFD was assumed sufficient to describe the required function, and the DFD was used intact.

Extracting complete and useful information from the users was the greatest challenge in building the required system model. Often, a desired capability which already existed in one form or another was expressed as a needed requirement. Also, users tended to talk in terms of specific physical desires rather than functional requirements. As in the existing system modeling process, several interpretations of the function had to be developed.

Model comparison was mainly an administrative task. The actual additional requirements were discovered as the required model was being created. Since the existing model was used as a baseline for the second model, the only items which influenced changes were those requirements not already existing, or existing and not wanted. A formal comparison was mainly used as a check to determine which of these discrepancies were valid, and which were a result of errors in the model or user misunderstanding.

The results of the model comparison pointed to the problems with the system, but not to the solutions. Development of a meaningful set of recommendations based on the

36

discovered discrepancies was an integral part of the evaluation, but was not well defined. Although software engineering disciplines exist to generate designs from the requirements, these techniques do not address the modification of several programs involved in one functional discrepancy. Recommendations were made in three categories based on personal observations of available resources, technical ability, and the political environment. These recommendations were directly developed to assist in problems discovered from the structured evaluation approach.

Recommendations

Structured evaluation can be a valuable tool for determining a system's ability to satisfy user requirements. The methodology described can be accomplished in a series of steps to discover problem areas. The following recommendations are suggested as additions or deviations from the described process:

1. Use more than one analyst as a check of interpretations and results.

2. Use the same functions for both models as much as possible. This aids in comparison and model generation.

3. Train the user in the modeling conventions, or use a technique he is already familiar with. This assists in model verification, and provides a useful document after the evaluation effort.

4. Carefully consider the level and validity of the defined functions prior to modeling to avoid major remodeling efforts.

The concepts of structured evaluation can be utilized in other areas of implementation. Some possible applications

37

are system familiarization and development verification.

Modeling is a powerful tool for learning a system.
It also provides a concise functional representation of a
complex system. These advantages suggest the possibility
of using this technique for gaining familiarity with an un-
known system not necessarily being evaluated. This effort
provides the analyst with a structure for learning the
system, as well as producing a useful piece of documentation.
The model can be used at a later date for analysis of modi-
fications, reference of system flow, and as a guideline for
other system documentation such as a user's guide.

Another possible application of structured evaluation
is as a tool for requirements definition and subsequent veri-
fication of a new system development. A normal requirement
model of a non-existing system can be created based on user
interviews and studies, as is normally done in structured
analysis. After the system is developed, a similar model
can be created of the developed system to check for discrep-
ancies with the specified requirements. This process is the
reverse of structured evaluation, but results in a similar
comparison. An advantage of this "reverse structured eval-
uation" is that the user interviews will not be influenced
by an existing system.

Structured evaluation is a viable software engineering
tool to analyze existing computer systems. Further study is
needed to develop a structured approach for generating pos-
sible recommendations from a list of model discrepancies.

38

However, even without such a methodology, the ability to define problem areas utilizing a structured approach enhances the completeness and accuracy of the results.

Bibliography

1. Alford, M. W. <u>A Requirements Engineering Methodology for Real-Time Processing Requirements</u>. Redondo Beach, California: TRW, September 1976.

2. Boehm, B. W. <u>Software Engineering</u>. Redondo Beach, California: TRW, October 1976.

3. Brooks, F. P. <u>The Mythical Man Month</u>. Reading, Massachusetts: Addison-Wesley Publishing Company, 1975.

4. De Marco, T. <u>Structured Analysis and System Specification</u>. New York: Yourdon, Inc. 1978.

5. DPI 6309-DAR-77H-61. Data Automation Request. March 1977.

6. Letter of Request for Purchase of an Intelligent Terminal, Attch 3. 14 April 1977.

7. PDS-4 System Reference Manual (ID 474721-0110). Needham, Massachusetts: IMLAC Corporation, August 1977.

8. Ross, D. T., et. al. "Software Engineering: Process, Principles, and Goals." <u>Computer,</u> <u>8</u>: 17-27 (May 1975).

9. Ross, D. T. and K. E. Schoman. "Structured Analysis for Requirements Definition." <u>IEEE Transactions on Software Engineering</u>, <u>SE-3</u>: 6-15 (January 1977).

10. SofTech, Inc. <u>An Introduction to SADT</u>. (SofTech Document #9022-78R). Waltham, Massachusetts, November 1976.

11. Teichroew, D. and E. A. Hershey. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems." <u>IEEE Transactions on Software Engineering</u>, <u>SE-3</u>: 41-48 (January 1977).

12. University of Michigan. <u>URL/URA Overview</u>. Prepared for US Air Force Electronic Systems Division Contract F19628-76-C-0197, February 1977.

13. Zelkowitz, M. V. "Perspectives on Software Engineering." <u>Computing Surveys</u>, <u>10</u>: 198-216 (June 1978).

Appendix A

Existing System Model

INLAC GRAPHICS SYSTEM
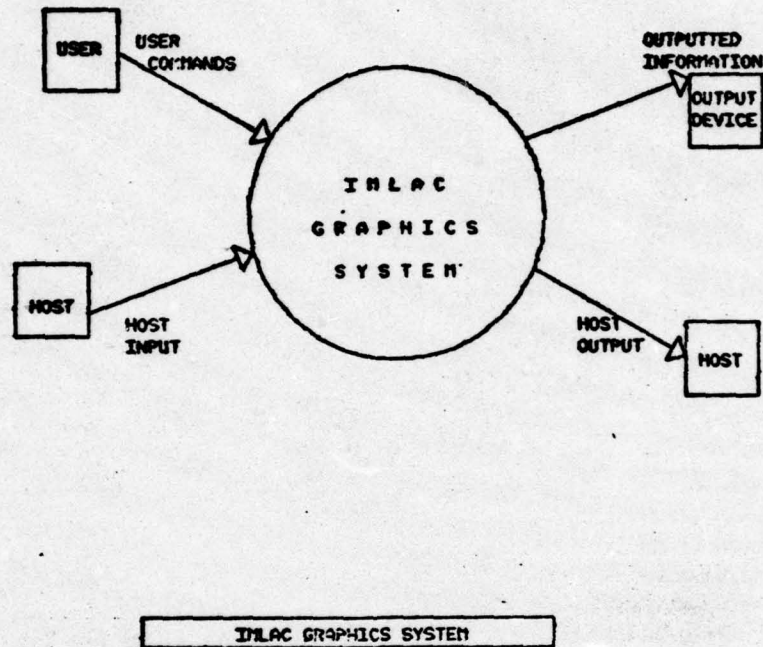
TEXT:

The IMLAC graphics system receives its input from either
a terminal user or a connected host computer. The user can
input commands through the keyboard, data switches, or light
pen. The host interfaces to the system through a communication
channel which is connected via a telephone modem. Output from
the system is transmitted back to the host, or to an output
peripheral such as the display screen or the printer.

```
              USER
             COMMAND                                    EXECUTING
                                                         PROGRAM

                        COMMAND      COMMAND     COMMAND
                         DATA         DATA         DATA
                                      FILE
                    1                              3

               INTERPRET                        EXECUTE
                COMMAND                           FILES


                        FILE
                        NAME
                    *            2
                                              LOADED
                             LOAD              FILE
                             FILE


                        FILE
               INLAC    DATA
               FILE
```
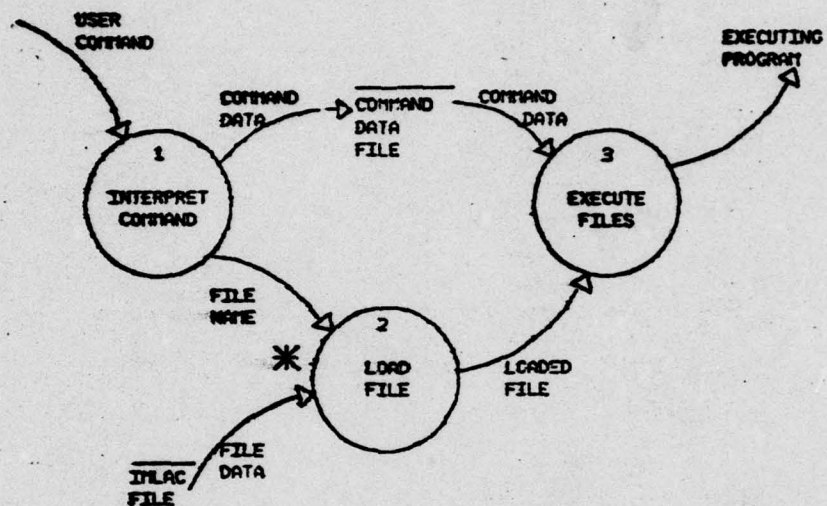
1.	OPERATING SYSTEM

TEXT:

The normal user command to the operating system consists of two portions: the name of the file to be executed, and some specified optional parameters. These parameters are written to a command file which is read by the program once loaded and put into execution. The file name is used to load the appropriate file off the disk.

```
                              DISPLAYED
                              INFORMATION
    DISPLAY                   ┌─────────┐              STORE          COMMAND
    SYSTEM                    │ DISPLAY │              COMMAND         DATA
  INFORMATION                 └─────────┘               DATA          FILE

                                              COMMAND
                                               DATA
    SYSTEM                                              COLLECT    ⊕
  INFORMATION                                            INPUT
   ┌────────┐                                                          SYSTEM
   │ SYSTEM │                   KEYBOARD                              FILE NAME
   │ TABLES │                    INPUT              FILE
   └────────┘                                       NAME
                        READ                                  CHECK     ⊕
     USER             KEYBOARD                                SYSTEM
    INPUT                                                     FILES
                                                                       USER
                                                                     FILE NAME
```
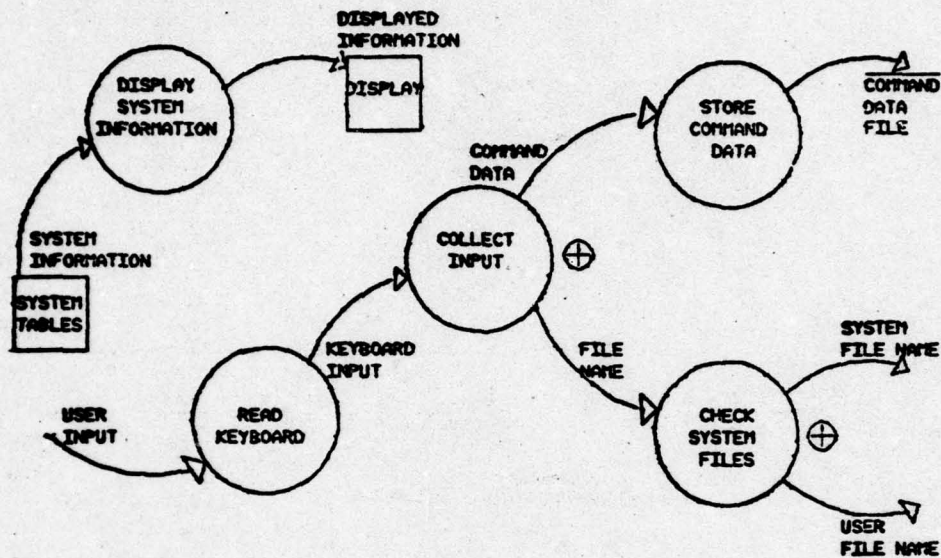
```
┌──────────────────────────────────────┐
│  1.1    INTERPRET COMMAND             │
└──────────────────────────────────────┘
```

TEXT:

Two basic functions are performed by the interpreting
program of the IMLAC: accepting user input, and displaying
system information. The information is located in system
tables which are built from data stored on disk and user
supplied data. User input is broken into the two categories
described in the operating system text. The file name portion
is further defined as either a system or user file. This
breakout is necessary to create the correct name of the file
to be loaded.

43

```
                                    ERROR
                                    MSG        ▷ ERROR
                                                 FILE
                        WRITE
        ERROR          ERROR
        MESSAGE        TO FILE
FILE                                      ADDRESS
NAME                                      ERROR          ⊕
                                                    STORE
        SEARCH      ⊕        ⊕        ⊕   IN CORRECT
     ✱  DIRECTORY                             CORE
                                            LOCATION      LOADED
                                                          PROGRAM
        DIRECTORY            ERROR
                            MESSAGE
   DIRECTORY                                   PROGRAM
   ENTRY                                       FILE
IMLAC
DIRECTORY                              ⊕
FILE                    CHECK        VALID      READ
                        USER         USER       FILE
                     ELIGIBILITY
                                        ✱
                                        ▷
                                    FILE
                                    DATA
                                IMLAC
                                FILE
```
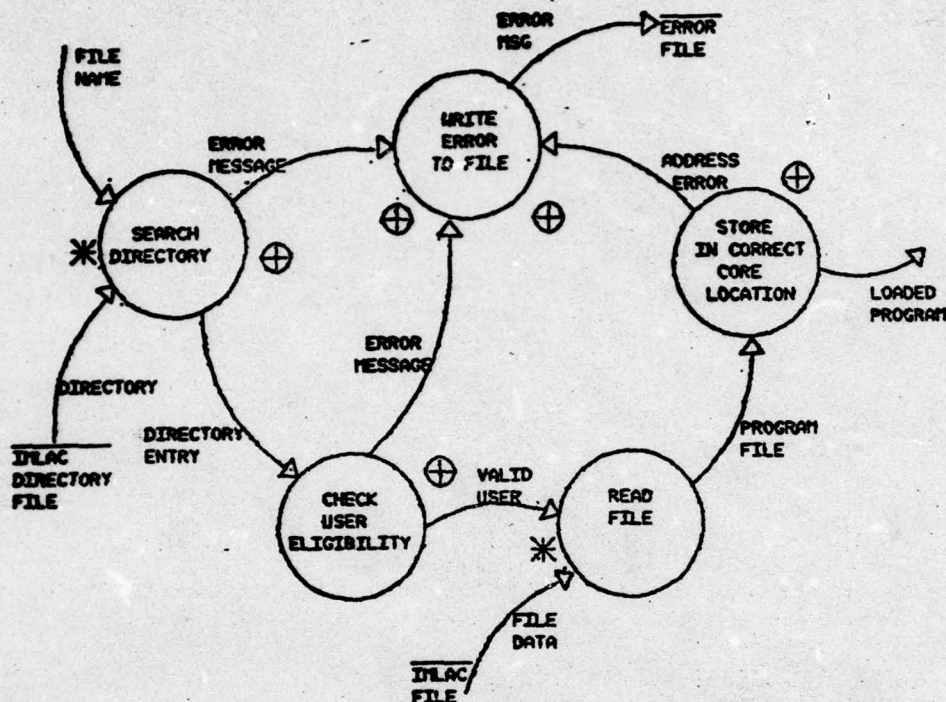
1.2	LOAD FILE

TEXT:

The file name input is either the system or user file
name generated from the command interpretation. The file's
existence is verified using the disk directory which resides
on a separate IMLAC file. The appropriate entry in the direc-
tory is used to validate that the user has the required per-
mission to use the file, and to determine the physical location
of the file on the disk. Once read, the file is loaded into
the correct locations in memory. All errors are written to
an error file for use by the operating system.

44

```
                    ┌──────────┐
                    │ PROGRAM  │
                    │ FILE     │
                    └──────────┘
                         │ FILE INFORMATION
                         ↓
                    ╭──────────╮          STARTING           ╭──────────╮
                    │ DETERMINE│          ADDRESS            │ TRANSFER │        EXECUTING
              ⊕     │  START   │ ──────────────────────────→ │ CONTROL  │ ─────→ PROGRAM
                    │ ADDRESS  │                      *      │          │
                    ╰──────────╯                             ╰──────────╯
                         ↑                                        ↑
              LOAD                                        LOADED
              COMMAND                                     PROGRAM
```
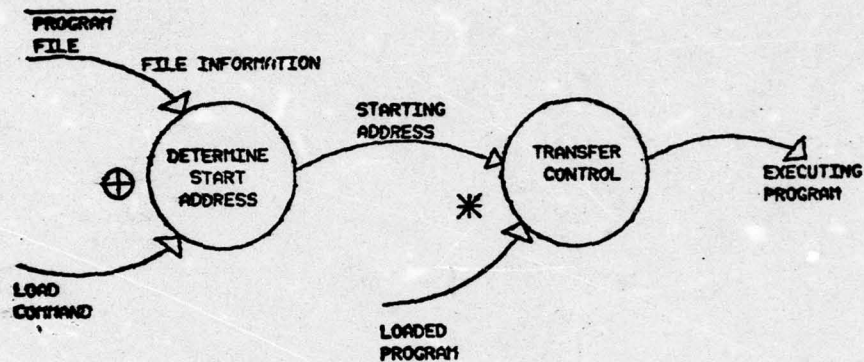
1.3	EXECUTE FILE

TEXT:

File execution requires the determination of the starting
address in memory. This is found from information in the file
which contained the program or from user supplied data. Actual
execution simply involves placing the starting address as the
current executing address.

45
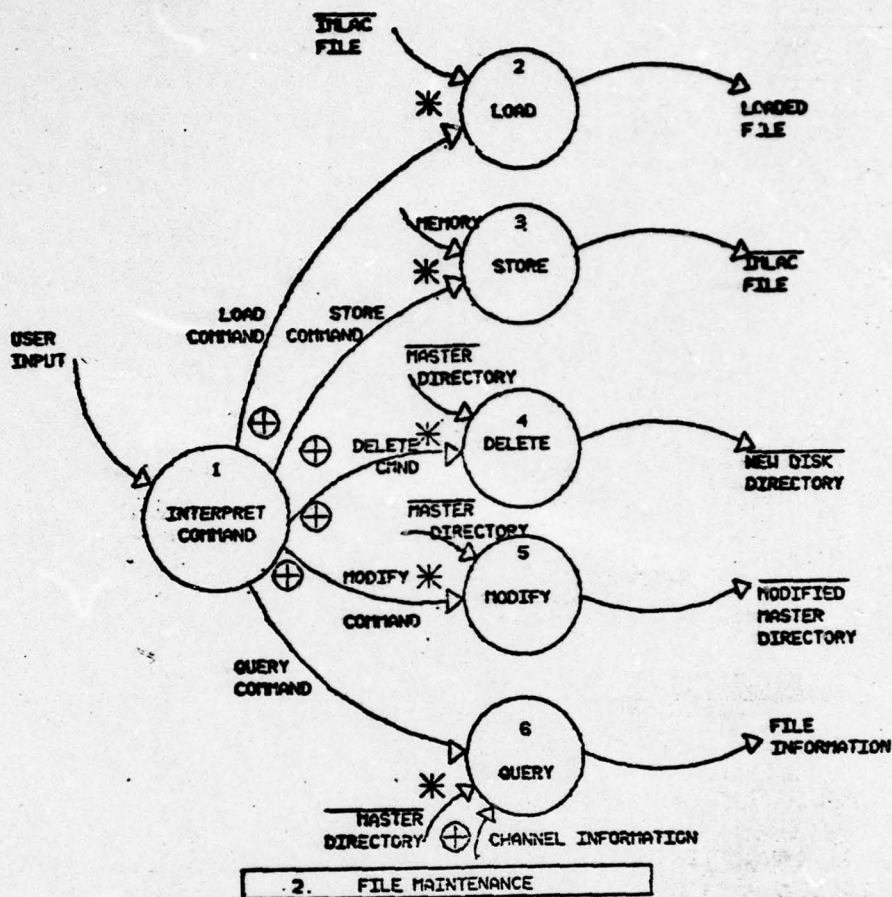
2. FILE MAINTENANCE

TEXT:

File maintenance is divided into five logical functions
as specified in the diagram. Each separate function requires
the user command (which includes the command file), and some
other source of information. The file information output is
directed to an output device for user viewing.
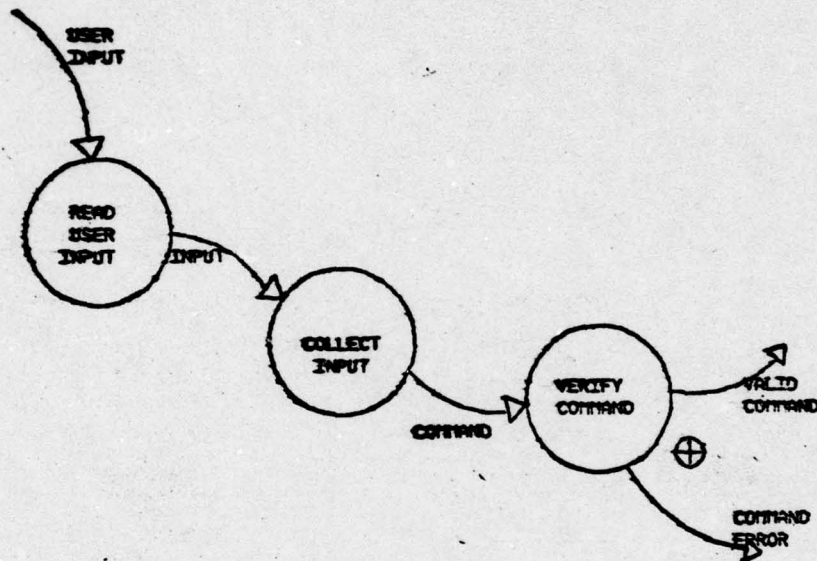
```
┌─────────┬──────────────────────┐
│  2.1    │  INTERPRET COMMAND   │
└─────────┴──────────────────────┘
```

TEXT:

Command interpretation is a straight forward task of reading user input, collecting it into a usable format, and verifying its validity.
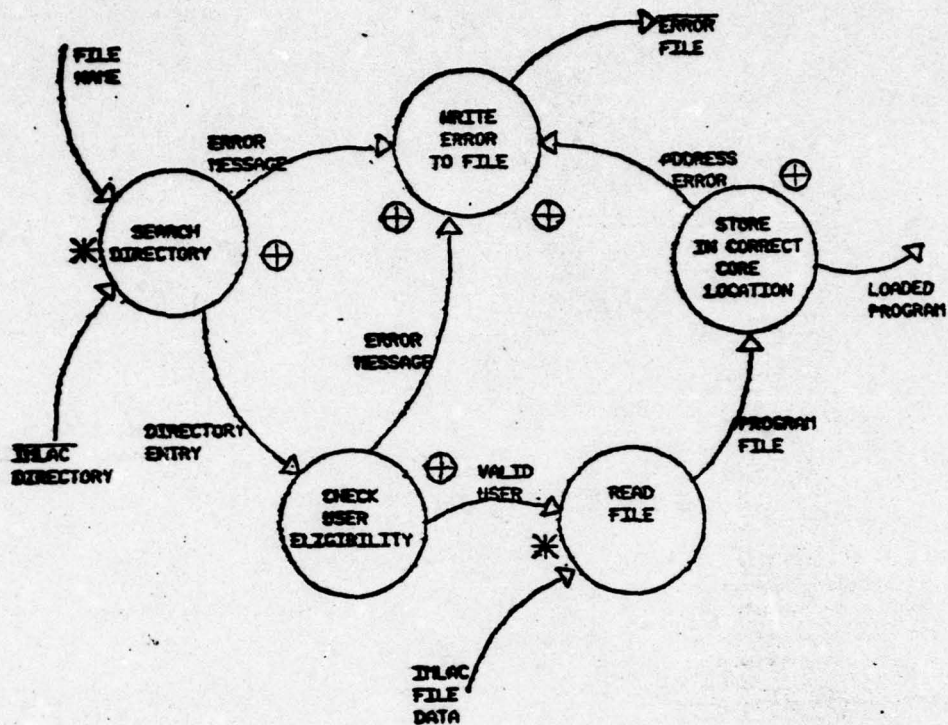
2.2   LOAD FILE

TEXT:

The process of loading a file is identical to the same process described in diagram 1.2. It is included here for model completeness.

48

```
                    STORE COMMAND

              DECIPHER
              COMMAND           CHECK            VALID
                             PERMISSION       PERMISSION    DELETE      DELETED
                                                             FILE      FILE NAME
              *
         FILE NAME                                                               UPDATE
                                          NEW FILE                            MASTER FILE
                                            NAME                               DIRECTORY
                             EXISTING                                                       NEW
                               FILE                                                         MFD
              CHECK                                         NEW                               MFD
            DIRECTORY  ⊕              ⊕                   FILE NAME
                                                 CREATE    *
                                                  FILE
                             NEW FILE                               CREATED              WRITTEN
                                                                     FILE                 FILE

                                                                         WRITE
                                                                         FILE
            MEMORY

                      *       FORMAT                                *
                               FILE
                            INFORMATION
            PARAMETERS                        FILE INFORMATION
```
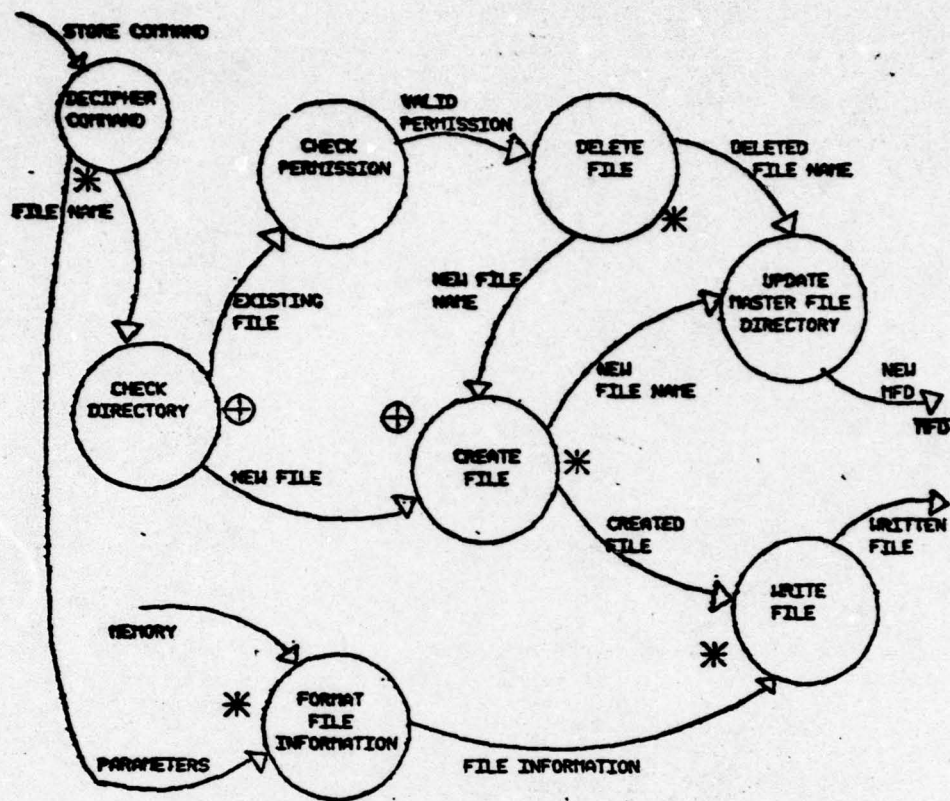
2.3	STORE FILE

TEXT:

File storage requires several checks to be accomplished prior to the actual writing to the disk. If the file is found to already exist by the directory search, and the user has the proper permissions, the existing file must be deleted and a new file created before the loaded file is stored. If the file is new, the file is created directly. The file creation involves updating the system tables and making the space available. The new file name is then entered into the file directory. Formatting the file for writing may require information provided by the user.
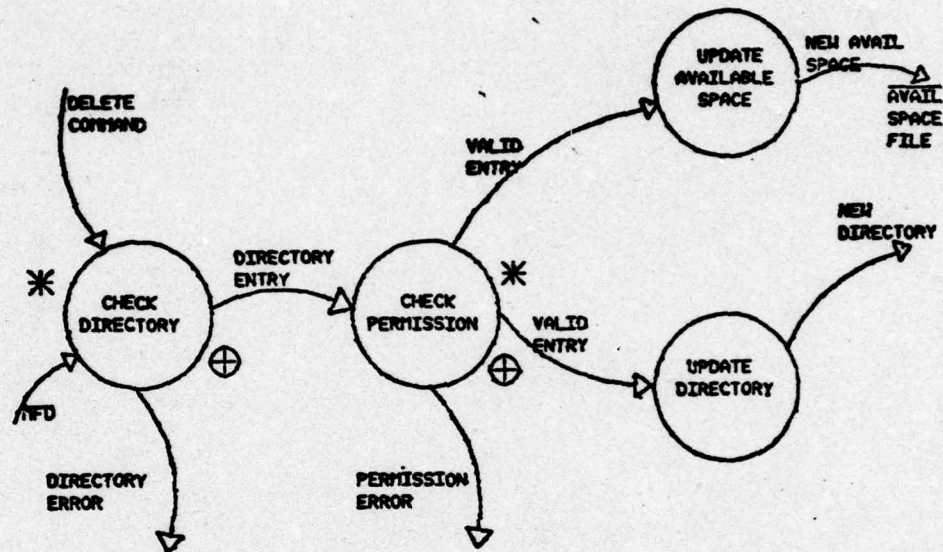
2.4   DELETE FILE

TEXT:

Deleting a file requires the existence of the file and the permission of the user to modify it. Once eliminated, the file information is used to update the appropriate system tables and directory.

```
                    MODIFY
                    COMMAND

                              VALID
              CHECK          MODIFICATION          MODIFY
           MODIFICATION                  *         ENTRY      MODIFIED
                                                              DIRECTORY
                        *                                                    NEW
                                                                          DIRECTORY
           FILE                                 VALID
           NAME                                 ENTRY                    STORE
                                  CHECK                                 MODIFIED
                               PERMISSION    ⊕                          DIRECTORY
                  VALID
                  ENTRY

              CHECK      ⊕
           DIRECTORY                          PERMISSION
        *                                       ERROR

                          DIRECTORY
                            ERROR
         MASTER
        DIRECTORY
```

| 2.5 | MODIFY FILE |
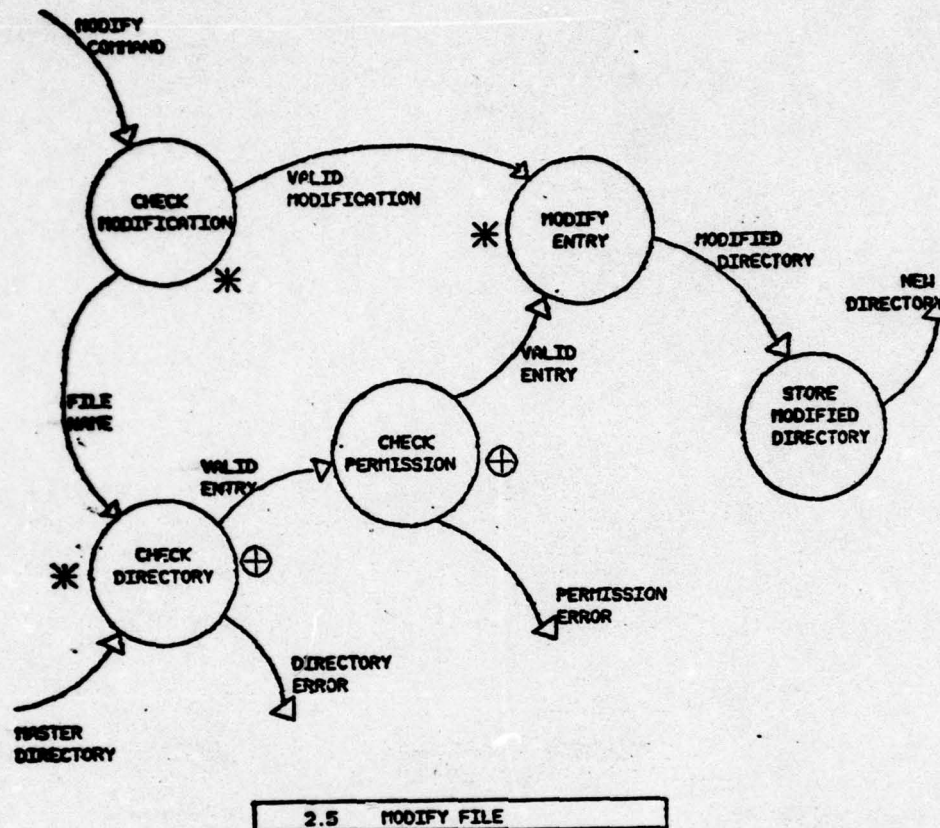
TEXT:

The modify command must be validated to ensure only authorized changes are made. The standard directory and permission checks are made for the file. The actual modification is only on the directory entry, such as a name change, or permission update. The new entry is written to the directory.

51

MASTER FILE DIRECTORY → SEARCH MFD

FILE ENTRY → EXTRACT INFORMATION

PRINTER OUTPUT

QUERY COMMAND

FILE QUERY

PRINT INFORMATION

QUERY INFORMATION

MFD

DIRECTORY

EXTRACT MFD INFORMATION

QUERY INFO → FORMAT QUERY INFORMATION

FORMATTED PRINTER INFORMATION

INTERPRET QUERY

QUERY

INFORMATION TABLE QUERY

FORMATTED DISPLAY INFO

DISPLAY INFORMATION

QUERY INFO

QUERY COMMAND

EXTRACT TABLE INFORMATION

INFORMATION TABLE

DISPLAYED OUTPUT

| 2.6 | QUERY FILE INFORMATION |

TEXT:

   Similar to file modification, the querying of a file
only involves the appropriate directory entry. The query is
divided into three types: on a specific file, on the entire
directory, or about system information. For the file request,
the information is derived from the directory entry and formatted
for output. The directory and information queries require
either the entire directory or system information tables to
extract the requested data. The formatted output is sent to
the printer or display depending on the command issued by
the user.

52
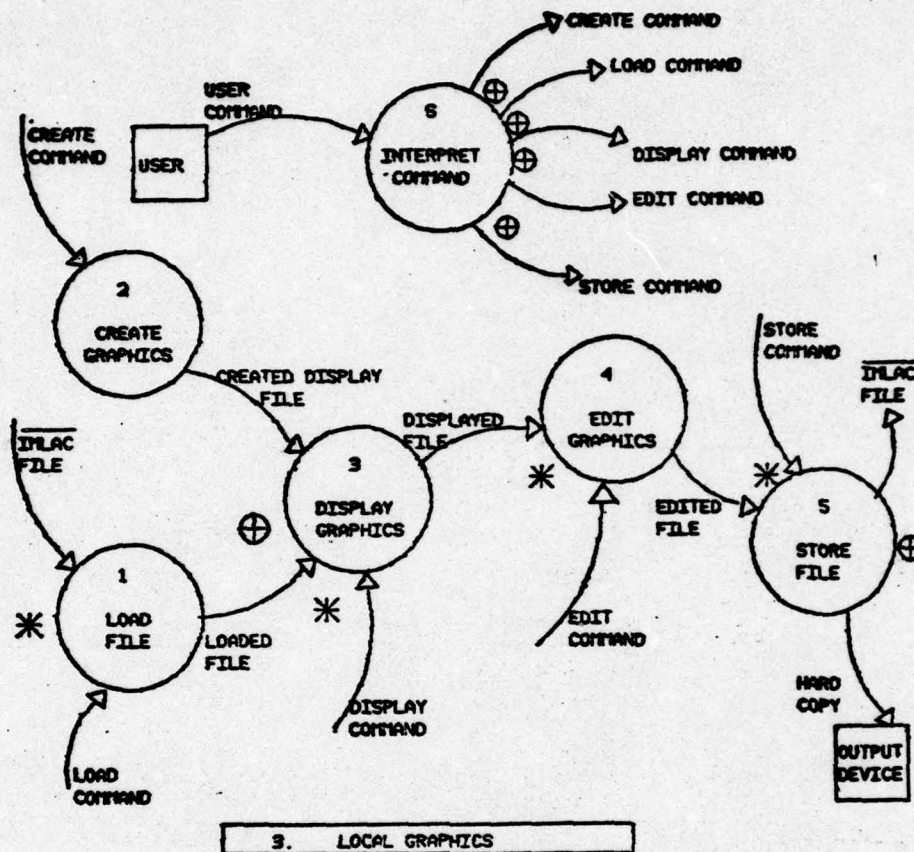
3. LOCAL GRAPHICS

TEXT:

Local graphics is the process of generating displays
using only the user input and the IMLAC computer. These dis-
plays can be created, displayed, and edited. To provide for
continuous use, the files can also be stored to and loaded
from a disk file. A hardcopy of the display is also available.

53
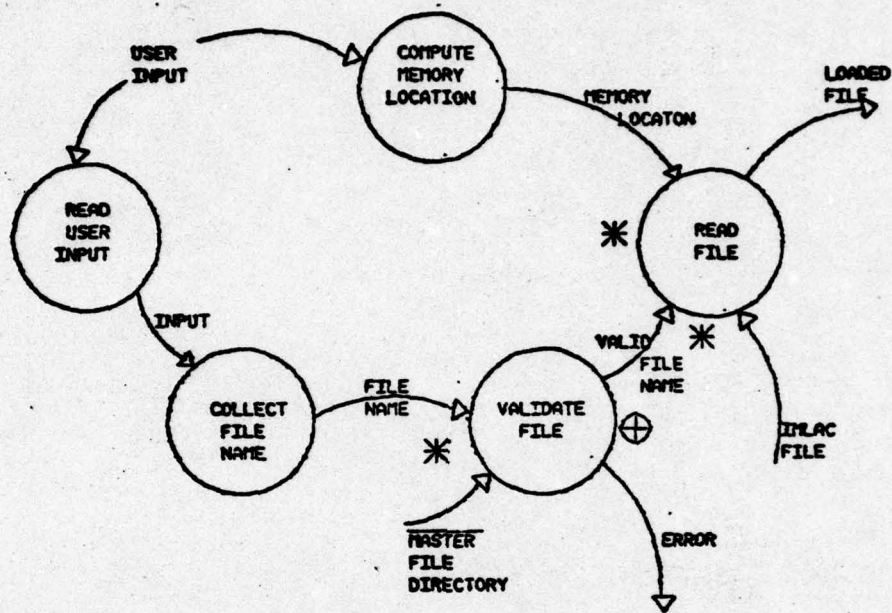
```
3.1    LOAD GRAPHIC FILE
```

TEXT:

Loading a graphics file is similar to other file loadings. The difference is that the user must specify the memory location to read the file into.

54
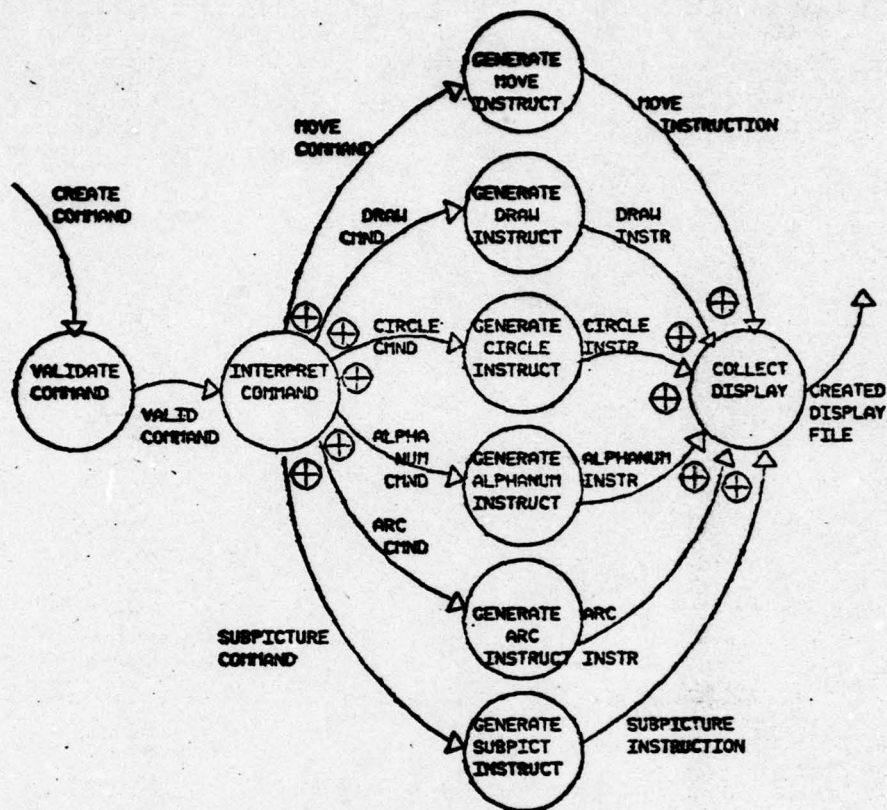
3.2   CREATE GRAPHICS FILE

TEXT:

To create a graphics file, the user command is interpreted as one of six display instruction types shown is the diagram. The appropriate display instructions are generated and collected to produce the created file. Internal to each instruction generator, user interaction may be required to complete the amount of information needed for that type.

```
       LOADED
       DISPLAY
       FILE
                          UPDATE
                 ⊕       DISPLAY      DISPLAY        TURN               DISPLAYED
                          LIST         LIST         DISPLAY             FILE
                                              *    PROCESSOR
                                                      ON
       CREATED
       DISPLAY
       FILE                    TURN ON
                               COMMAND

                          INTERPRET    ⊕         TURN
                          COMMAND                DISPLAY
                                                PROCESSOR
                                                   OFF
                                  TURN OFF
       DISPLAY                    COMMAND
       COMMAND
```
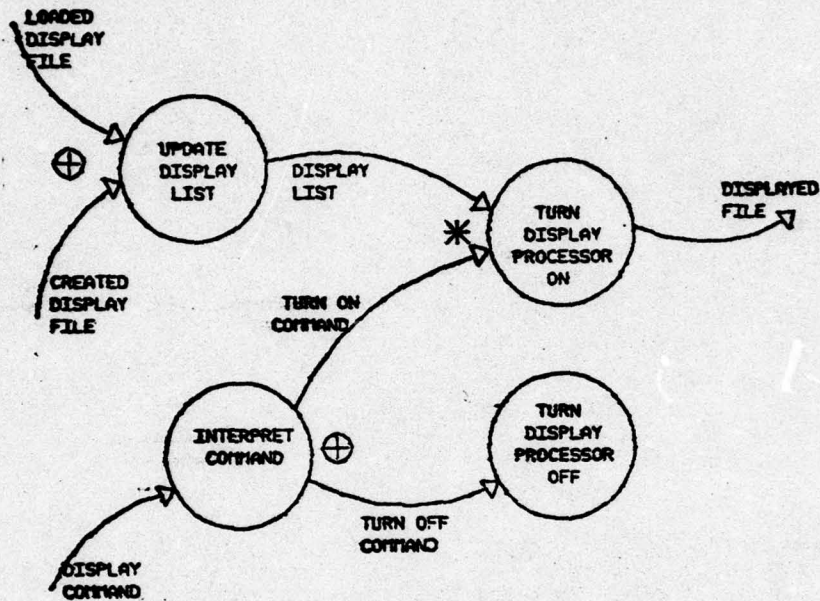
┌─────────────────────────────────┐
│ 3.3   DISPLAY GRAPHICS FILE   │
└─────────────────────────────────┘

TEXT:

Displaying a graphics file requires the master display
list be updated based on created or loaded files. The master
list is either executed by the display processor or not, de-
pending on the user command.

56
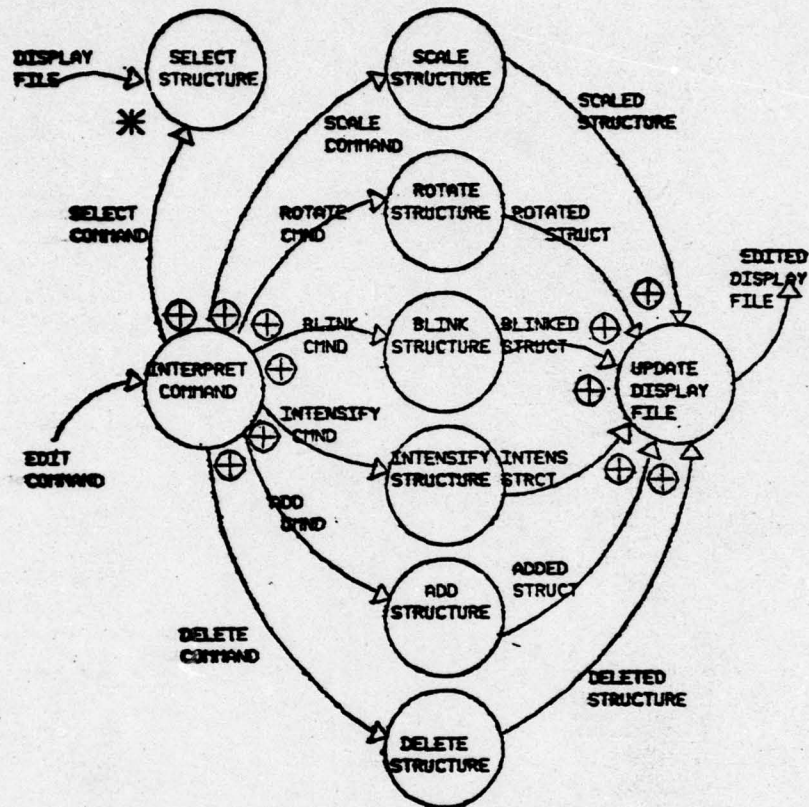
```
3.4    EDIT GRAPHICS FILE
```

TEXT:

Several types of graphic editing are available to the user. Each of these require that the desired portion of the display to be edited is specified by the user. The display instructions associated with this portion, or structure, are then modified to create the desired change.

EXISTING FILE
ERROR

CHECK
DIRECTORY ⊕

UPDATE
MFD

NEW FILE
ENTRY

AVAILABLE
ENTRY

CREATE
FILE ✳

NEW MFD

MFD

FILE
NAME

FILE
INFO

WRITE
LIST
TO DISK

STORE
COMMAND

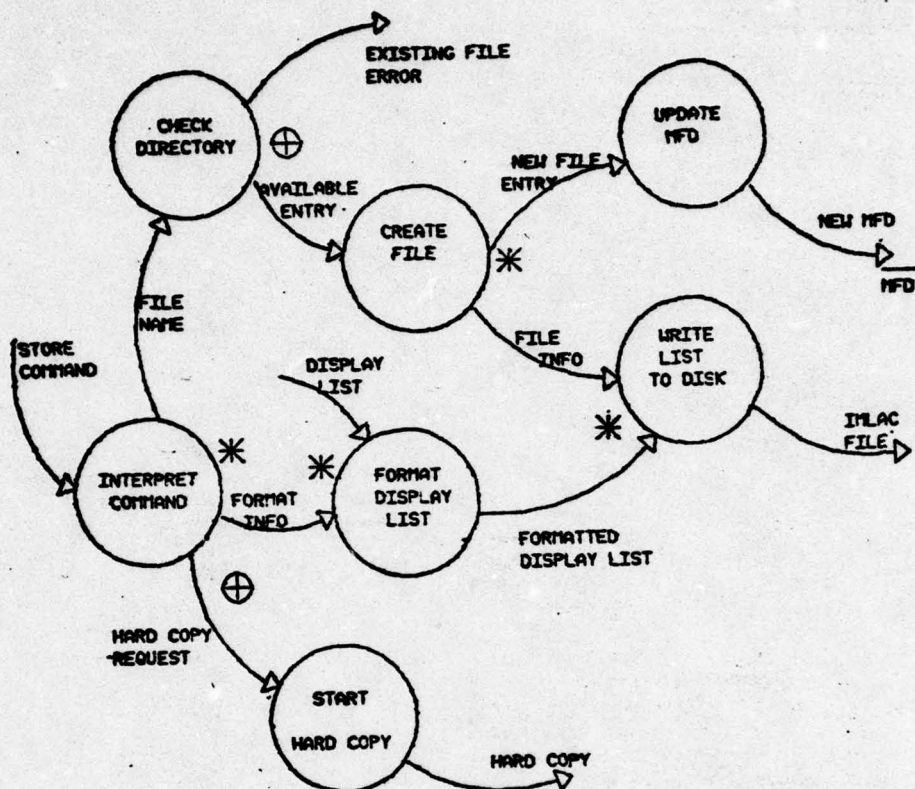DISPLAY
LIST

INTERPRET
COMMAND ✳ ✳
FORMAT
INFO

FORMAT
DISPLAY
LIST

IMLAC
FILE

✳

FORMATTED
DISPLAY LIST

⊕

HARD COPY
REQUEST

START
HARD COPY

HARD COPY

| 3.5 | STORE GRAPHICS FILE |

TEXT:

The output device for display storage must be inter-
preted as either disk or plotter. Storage to the disk is
similar to the process describe in 2.3 with the exception that
the file must not exist prior to the storage. Plotting the
display to the plotter is a hardware function initiated in
software.
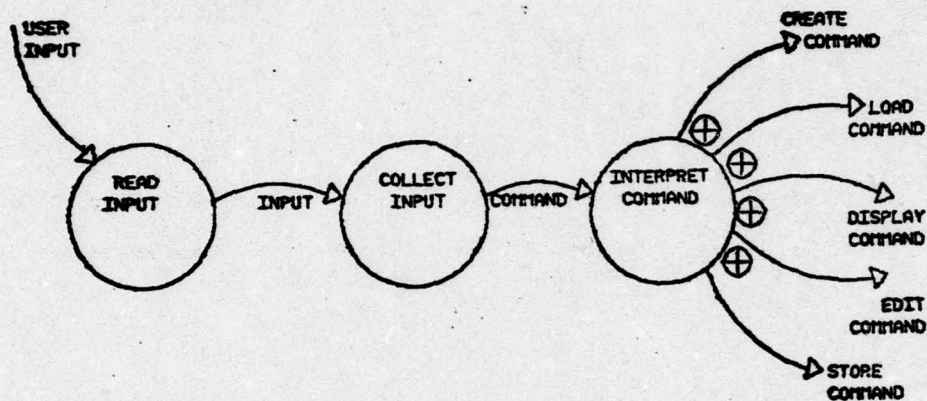
```
3.6    INTERPRET COMMAND
```

TEXT:

Command interpretation is a straight forward task of reading the user input, and interpreting its function. Types of user input are keyboard and light pen.

```
                              5
       ┌──────┐           ┌────────┐              ⊕──────▷ CREATE COMMAND
       │ HOST │   HOST    │INTERPRET│
       │      │──COMMAND──│ COMMAND │             ⊕──────▷ QUERY COMMAND
       └──────┘           └────────┘
                                                  ⊕──────▷ EDIT COMMAND

                                                  ──────▷ DISPLAY COMMAND
```
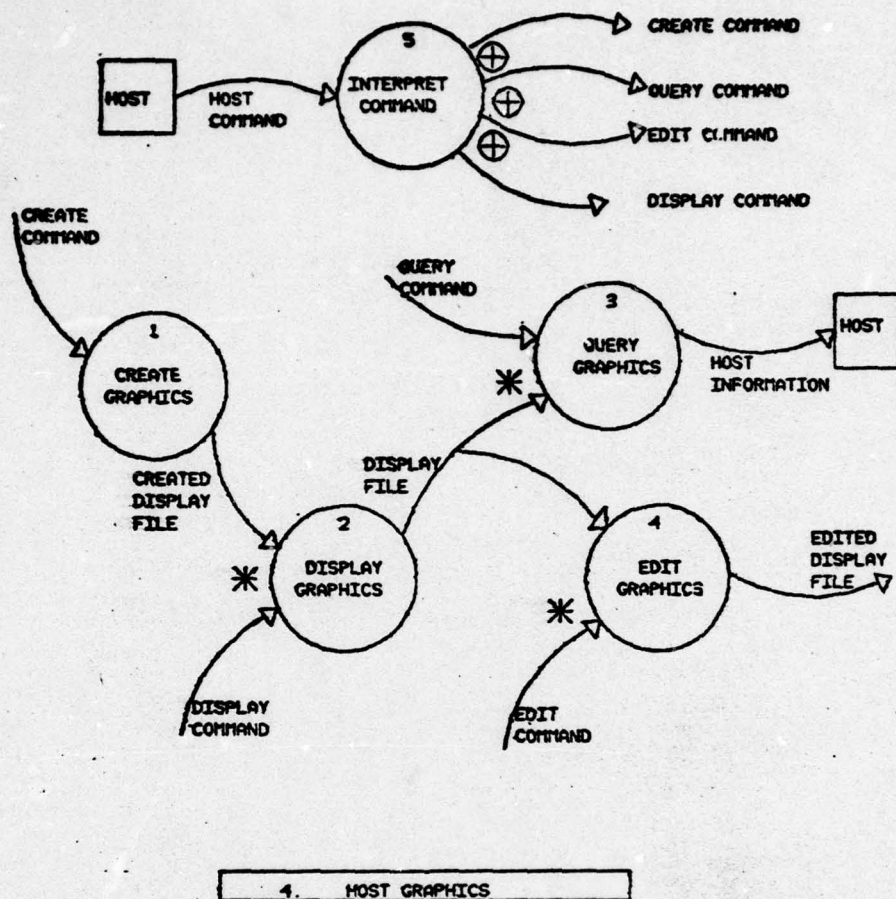
4.   HOST GRAPHICS

TEXT:

Host graphics is the process of developing displays
from commands issued by a connected host computer. The
functions performed are similar to local graphics except for
loading and storing complete displays and the ability to
query the graphics. Information obtained from these queries
is sent back to the host machine.

60

4.1 HOST CREATE GRAPHICS
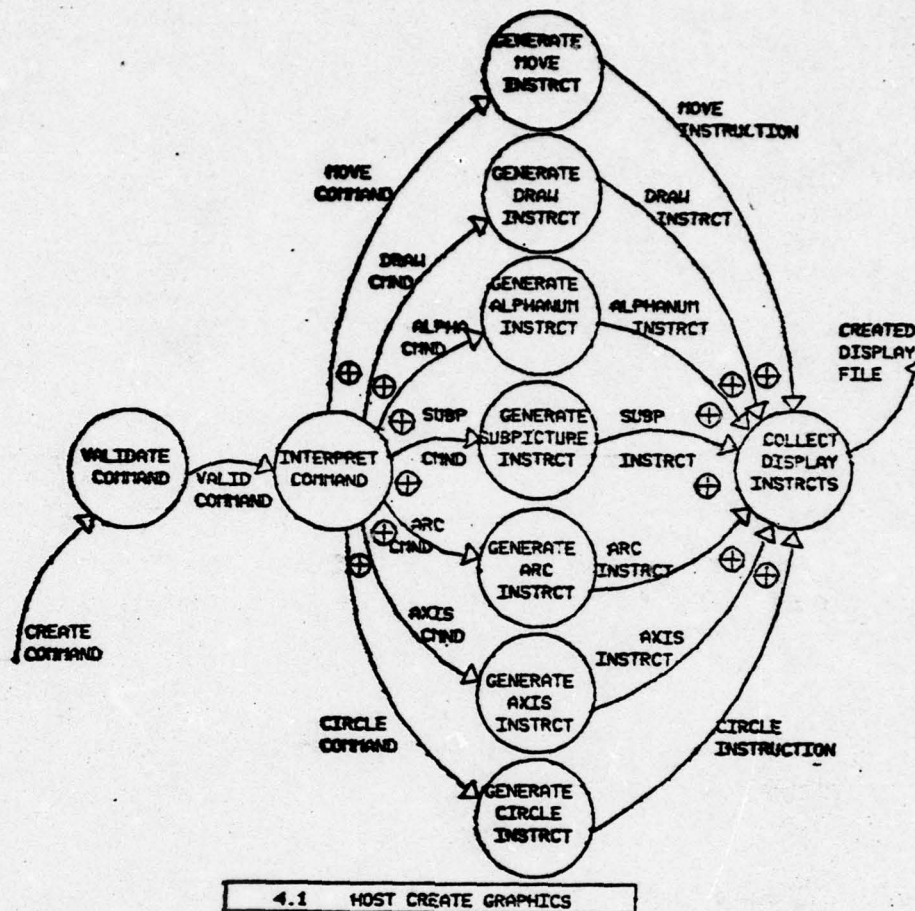
TEXT:

The creation process is similar to the same function in the local graphics diagram, 3.2. The only added capability is the axis command which draws an x,y axis.

61

LOADED DISPLAY
FILE

UPDATE
DISPLAY
LIST

DISPLAY
LIST

TURN DISPLAY
PROCESSOR
ON

DISPLAYED
FILE

CREATED
DISPLAY
FILE

TURN ON
COMMAND

INTERPRET
COMMAND

TURN DISPLAY
PROCESSOR
OFF

TURN OFF
COMMAND
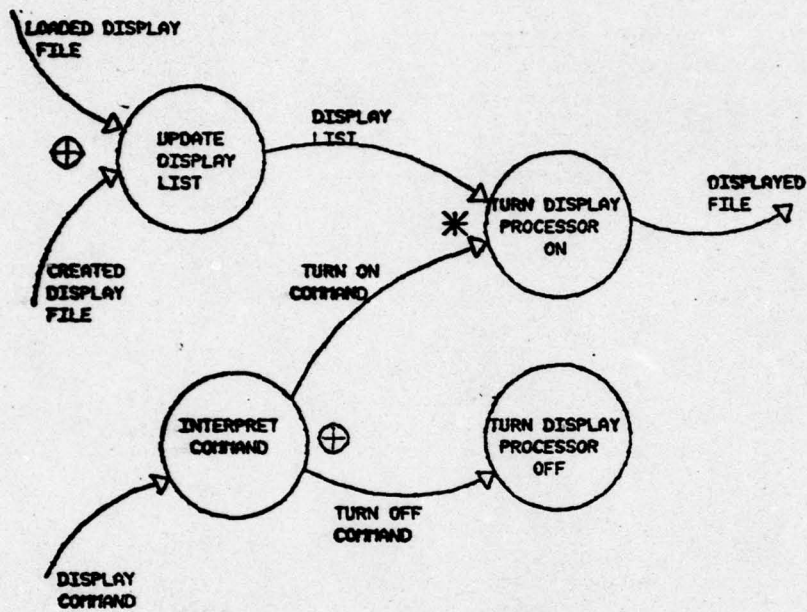
DISPLAY
COMMAND

| 4.2 | HOST DISPLAY GRAPHICS |

TEXT:

Displaying the graphics is identical to displaying
local graphics described in diagram 3.3. It is included
here for model completeness.

KEYBOARD REGISTER

READ KEYBOARD REGISTER

KEYBOARD CHARACTER

*

KEYBOARD COMMAND

READ DATA REGISTER

DATA REG CMND

DATA REG VALUE

HOST INFORMATION

INTERPRET COMMAND

LIGHT PEN COORD CMND

COMPUTE LIGHT PEN COORDINATES

LIGHT PEN COORDS

TRANSMIT INFORMATION TO HOST

QUERY COMMAND

LIGHT PEN SUBP CMND

COMPUTE SUBPICTURE INDEX

SUBP INDEX

CURSOR COMMAND

COMPUTE CURSOR COORDINATES

CURSOR COORDINATES

| 4.3 | HOST QUERY |

TEXT:

Allowing the user to supply information to the host from graphic queries is a central theme for interactive graphics. Utilizing the different means shown in the diagram, the user can provide the host with a source of input based on visual interpretation of data results.

63
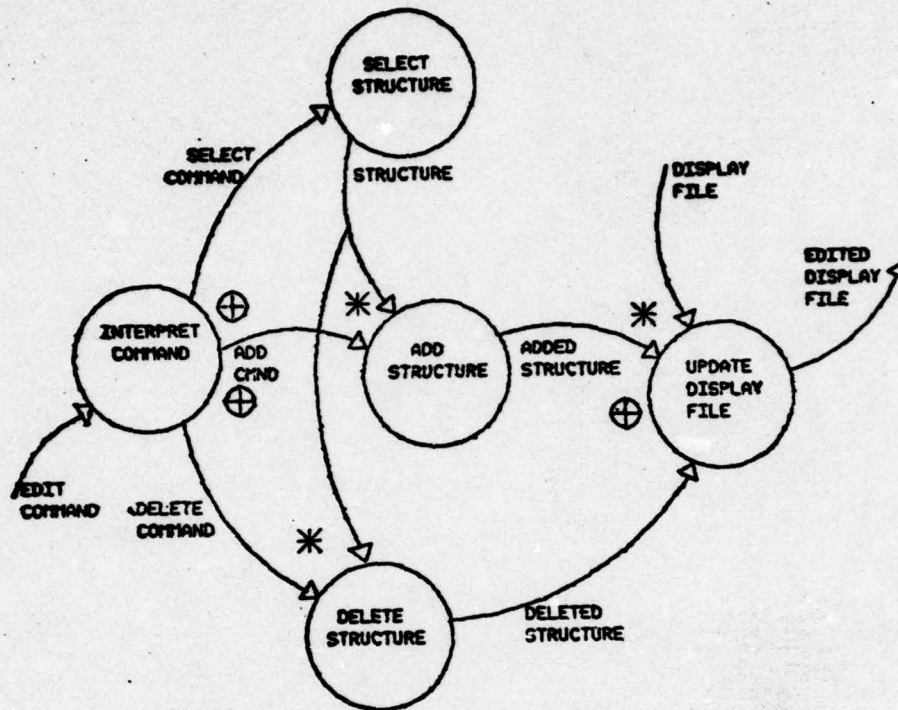
4.4   HOST EDIT GRAPHICS

TEXT:

Editing host graphics differs from the local capability.
The only functions provided are adding or deleting defined
subpictures. The other functions listed in diagram 3.4 must
be accomplished at the host with a new structure created.
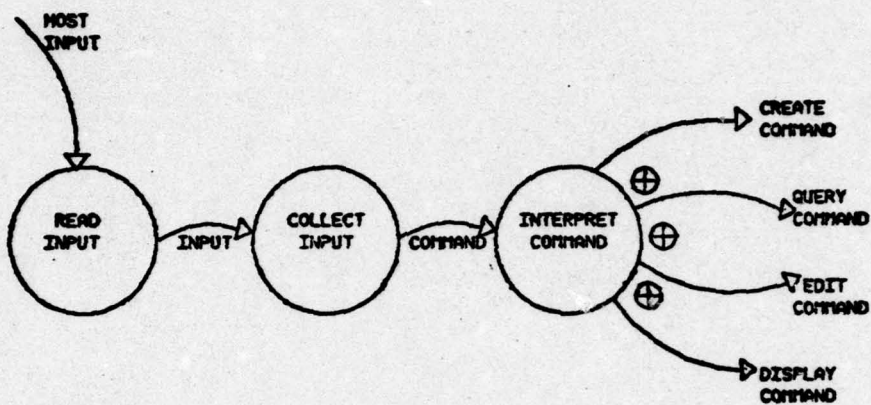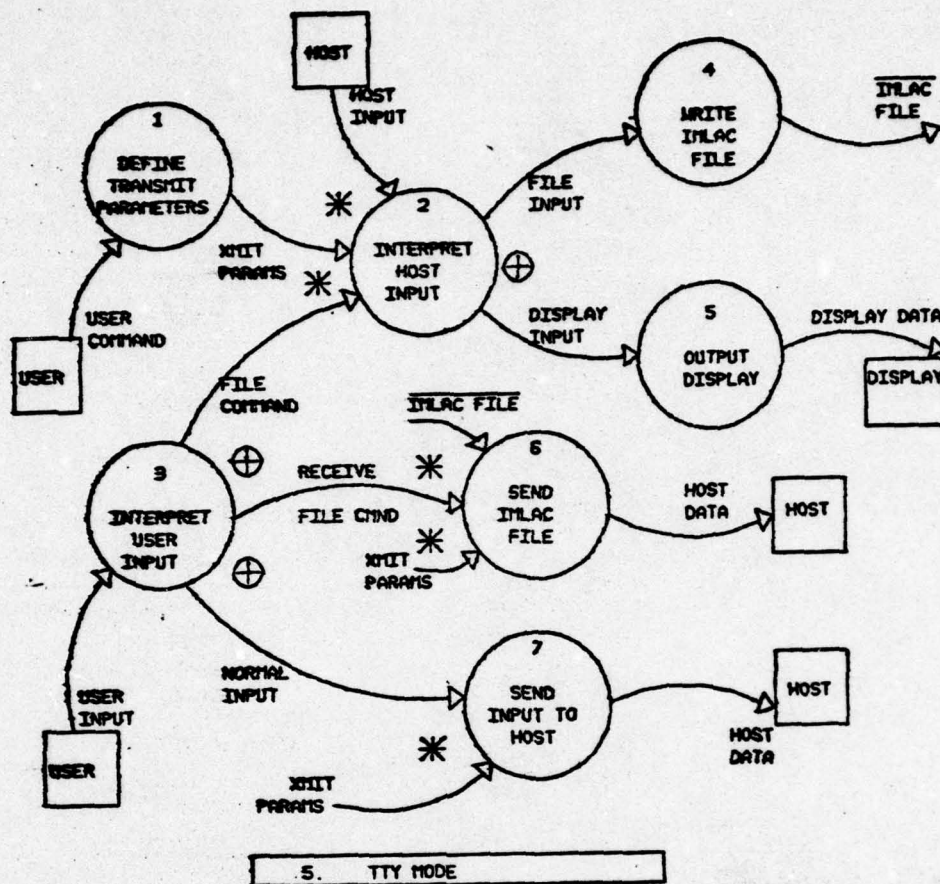
64

4.5 INTERPRET HOST COMMAND

TEXT:

Interpreting the host command is the same as for local graphics with a different breakout of command types.

65

```
                          ┌──────┐
                          │ HOST │
                          └──────┘
                           HOST
                           INPUT
        ╭─────────╮                                    ╭─────────╮
        │    1    │                                    │    4    │        ▽ IMLAC
        │ DEFINE  │                                    │ WRITE   │──────▷   FILE
        │TRANSMIT │                      FILE           │ IMLAC   │
        │PARAMETERS│        ╭─────────╮  INPUT          │ FILE    │
        ╰─────────╯    *    │    2    │─────────────▷   ╰─────────╯
             ▷     XMIT     │INTERPRET│ ⊕
         USER    PARAMS  *  │  HOST   │         ╭─────────╮
        COMMAND            │ INPUT   │ DISPLAY  │    5    │  DISPLAY DATA
   ┌──────┐                ╰─────────╯ INPUT    │ OUTPUT  │─────────▷
   │ USER │                           ─────────▷│ DISPLAY │          ┌────────┐
   └──────┘                                      ╰─────────╯         │DISPLAY │
              FILE                                                    └────────┘
             COMMAND            ▷ IMLAC FILE
                           ⊕       *    ╭─────────╮
        ╭─────────╮            RECEIVE   │    6    │    HOST
        │    3    │ ⊕         FILE CMND   │ SEND    │    DATA    ┌──────┐
        │INTERPRET│────────────────────▷ │ IMLAC   │─────────▷  │ HOST │
        │  USER   │           *  XMIT    │ FILE    │            └──────┘
        │ INPUT   │ ⊕           PARAMS ▷ ╰─────────╯
        ╰─────────╯
                      NORMAL              ╭─────────╮
                      INPUT               │    7    │           ┌──────┐
        ▷                  ─────────────▷ │ SEND    │──────────▷│ HOST │
     USER                                 │INPUT TO │            └──────┘
    INPUT               *                 │ HOST    │   HOST
  ┌──────┐           XMIT             ▷   ╰─────────╯   DATA
  │ USER │          PARAMS  ─────────────▷
  └──────┘
                  ┌─────────────────────────┐
                  │  5.    TTY MODE         │
                  └─────────────────────────┘
```

TEXT:

TTY mode is the means for communicating with a host time sharing system. Information from the host is interpreted using specified parameters from the user. These parameters define the type of communication, data format, and whether to display the input or put it directly to a file. The user input is separated into parameter definition, local commands, or normal input to be sent to the host. Using local commands, the user can send or receive entire files of data.
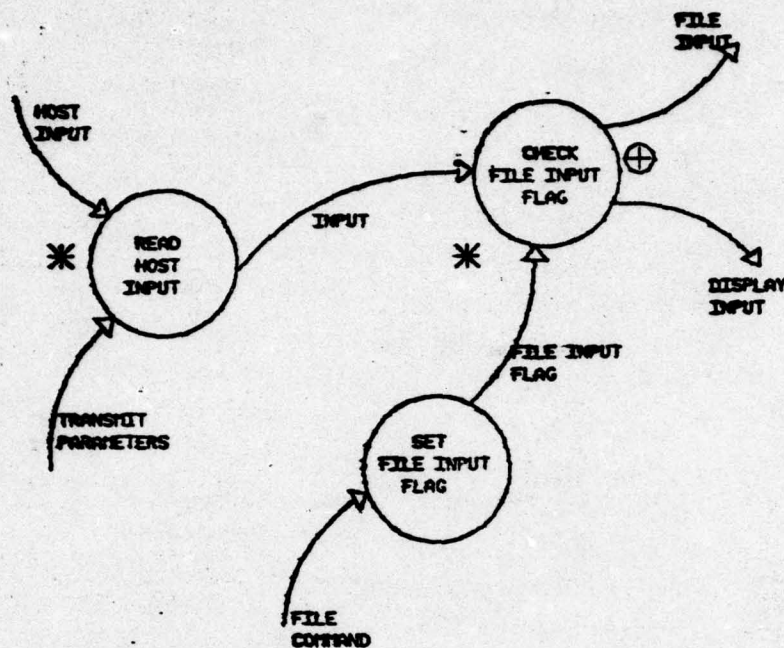
66
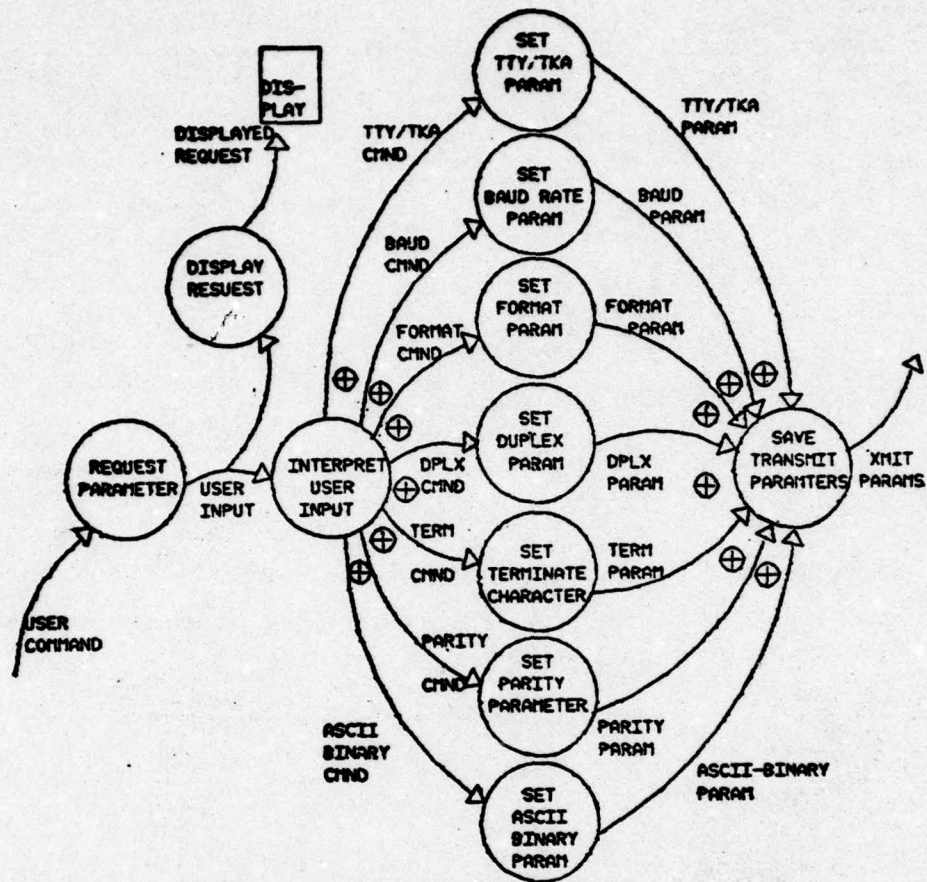
5.1 DEFINE TRANSMIT PARAMETERS

TEXT:

Several parameters can be defined by the user as illustrated in the diagram. These parameters are used to interpret different data formats and set transmitting characteristics. The requested parameter change is displayed for the user.
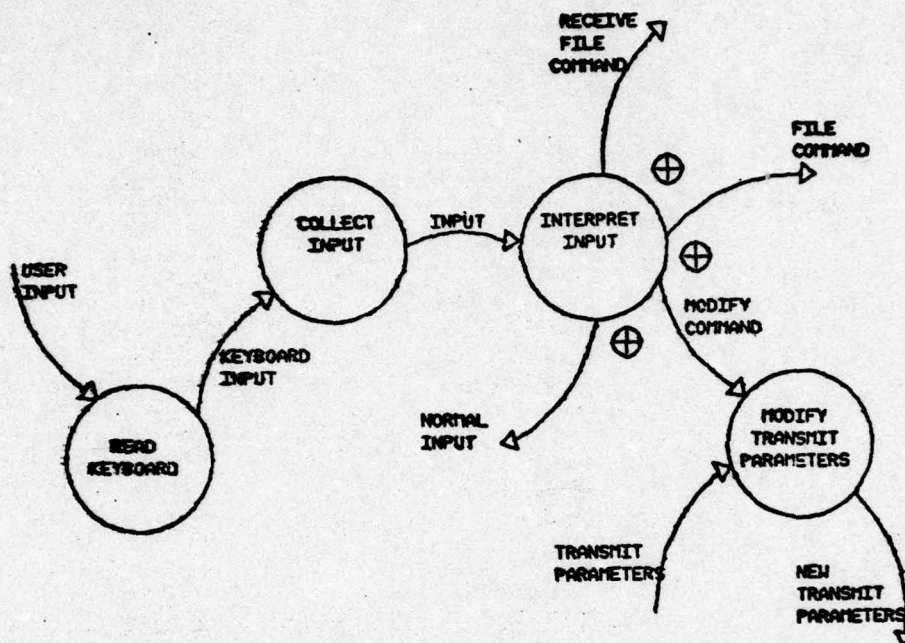
67

FILE
INPUT

HOST
INPUT

CHECK
FILE INPUT
FLAG

READ
HOST
INPUT

INPUT

DISPLAY
INPUT

FILE INPUT
FLAG

TRANSMIT
PARAMETERS

SET
FILE INPUT
FLAG

FILE
COMMAND

S.2	INTERPRET HOST INPUT

TEXT:

The host input is read and interpreted using the par-
ameters set by the user. Depending on whether the user
specified the incoming data as file information or not, the
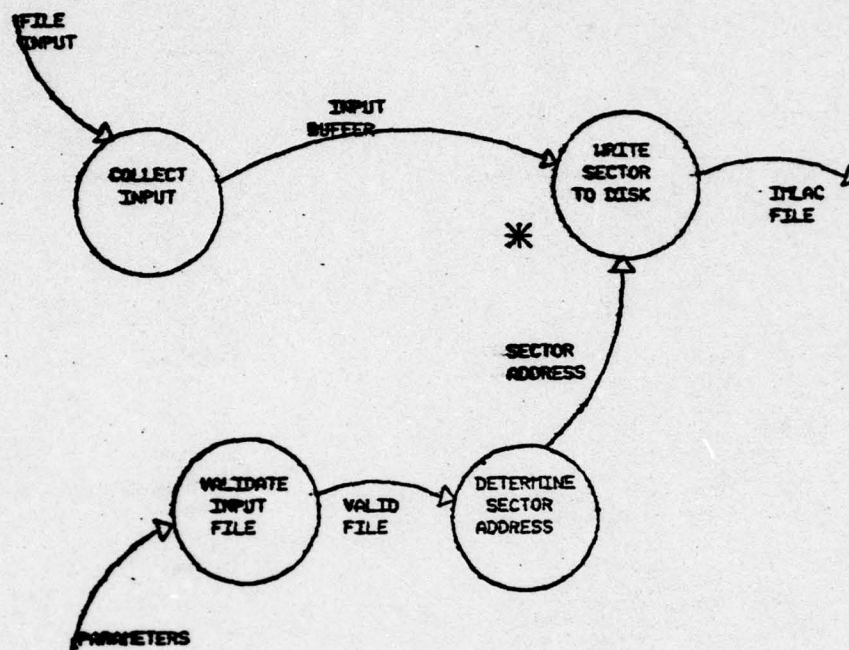input is either written into an IMLAC file or displayed on
the screen.

68

USER INPUT

READ KEYBOARD

KEYBOARD INPUT

COLLECT INPUT

INPUT

INTERPRET INPUT

RECEIVE FILE COMMAND

FILE COMMAND

MODIFY COMMAND

NORMAL INPUT

MODIFY TRANSMIT PARAMETERS

TRANSMIT PARAMETERS

NEW TRANSMIT PARAMETERS

5.3   INTERPRET USER INPUT

TEXT:

The user input is interpreted as one of four types.
Normal input is correctly formatted and sent to the host.
Two types of file commands are recognized to send and receive
entire data files. The fourth input type is to modify the
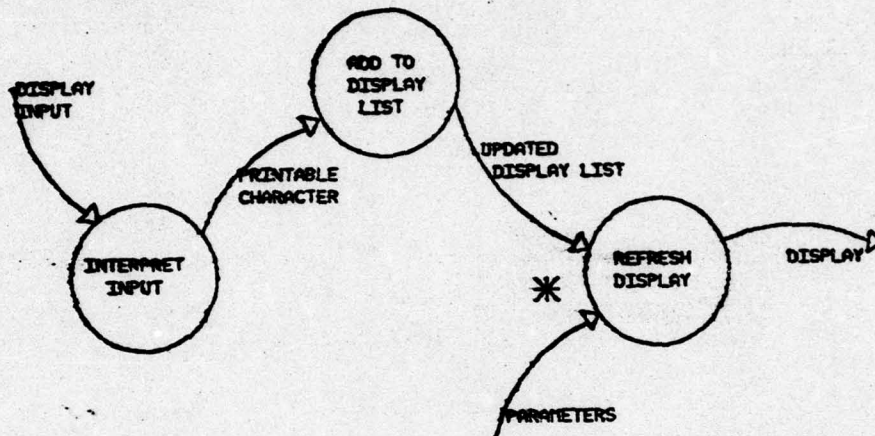user specified parameters.

69

FILE
INPUT

INPUT
BUFFER

COLLECT
INPUT

WRITE
SECTOR
TO DISK

IMLAC
FILE

*

SECTOR
ADDRESS

VALIDATE
INPUT
FILE

VALID
FILE

DETERMINE
SECTOR
ADDRESS

PARAMETERS

S.4   WRITE IMLAC FILE

TEXT:

Host information destined for an IMLAC file is collected
into buffers the size of a disk sector. The correct address
for writing these sectors must be determined from the par-
ameters set by the user.

```
                    ADD TO
                    DISPLAY
                     LIST

DISPLAY
INPUT                          UPDATED
                               DISPLAY LIST
              PRINTABLE
              CHARACTER
                                           REFRESH        DISPLAY
          INTERPRET                    *   DISPLAY
          INPUT


                                    PARAMETERS
```

5.5	OUTPUT DISPLAY

TEXT:

Host information which is not written to a file is displayed on the CRT. The input must be interpreted to be printable and defined as display instructions to add to the existing display list. The display is constantly refreshed to remain visible.
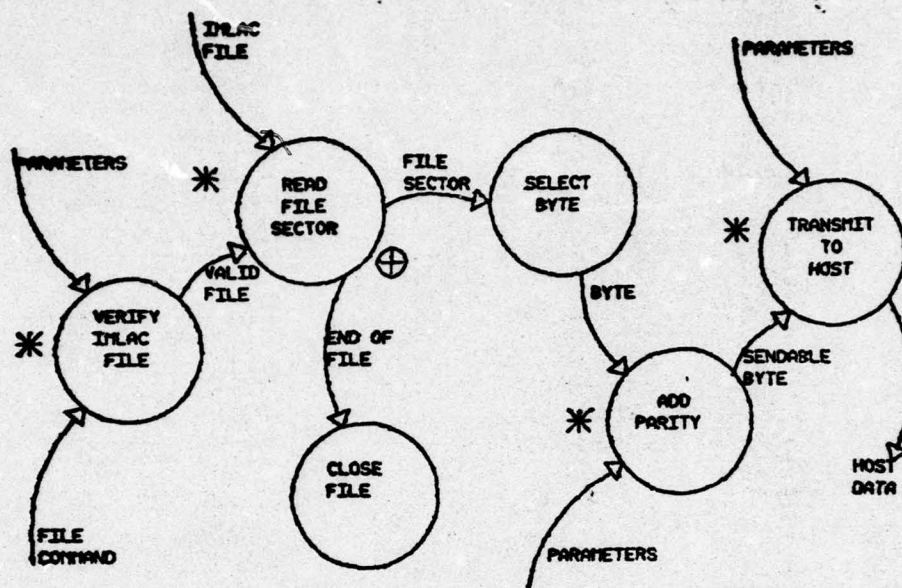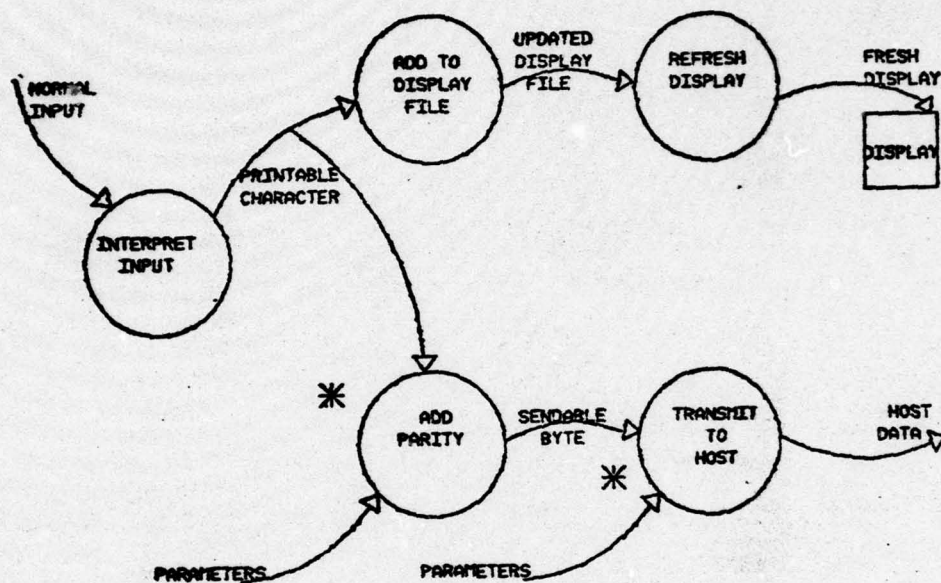
5.6   SEND IMLAC FILE

TEXT:

When an IMLAC file is sent to the host, the user's
permission to send the file must be validated. The file is
then read into memory a sector at a time. The sector is
transmitted a byte at a time after proper formatting.

```
                              UPDATED
                    ADD TO    DISPLAY     REFRESH        FRESH
                    DISPLAY   FILE        DISPLAY        DISPLAY
    NORMAL          FILE
    INPUT                                                   ┌─────────┐
                PRINTABLE                                   │ DISPLAY │
                CHARACTER                                   └─────────┘

      INTERPRET
      INPUT

              *       ADD      SENDABLE    TRANSMIT        HOST
                      PARITY   BYTE        TO              DATA
                                     *     HOST

              PARAMETERS          PARAMETERS
```

5.7	SEND INPUT TO HOST

TEXT:

   Normal user input is displayed on the screen along
with being transmitted to the host. The input must be for-
matted correctly before transmission.
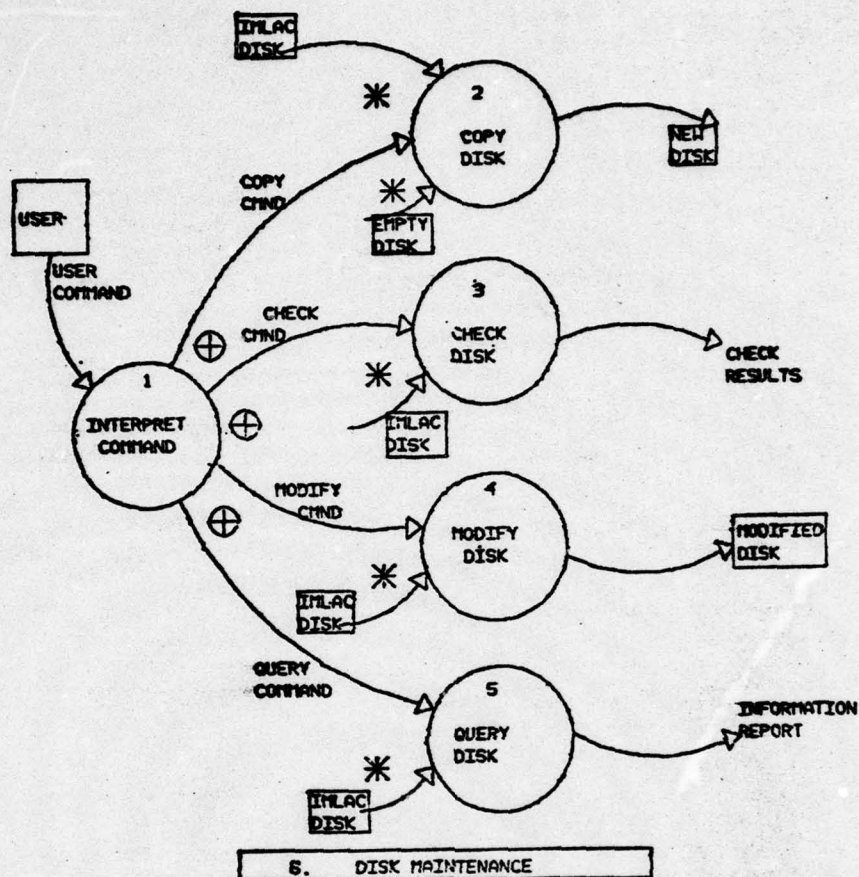
6.   DISK MAINTENANCE

TEXT:

     Disk maintenance is divided into the four functions
shown in the diagram. Physical disk cartridges used as
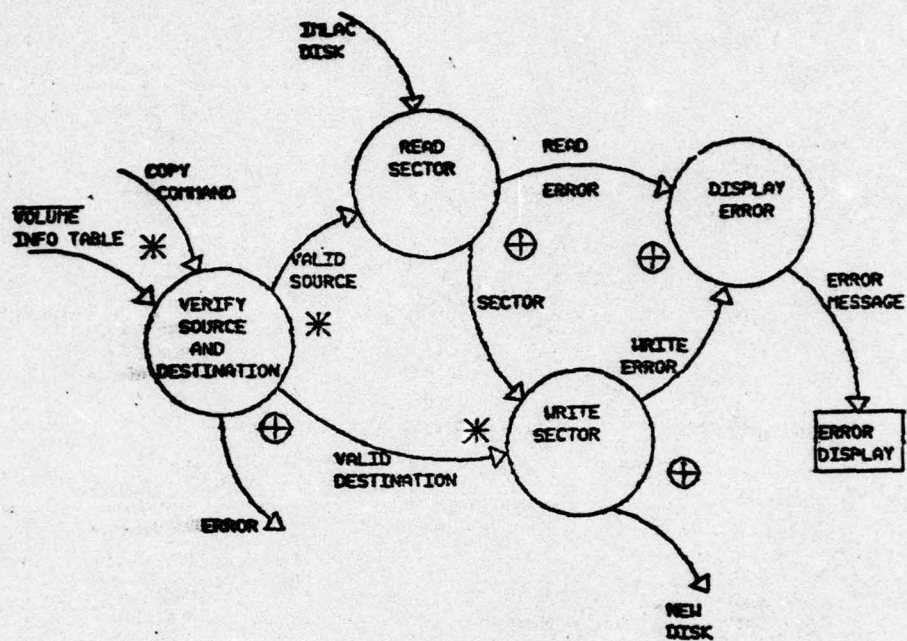input or output are shown as a source or sink for the data.
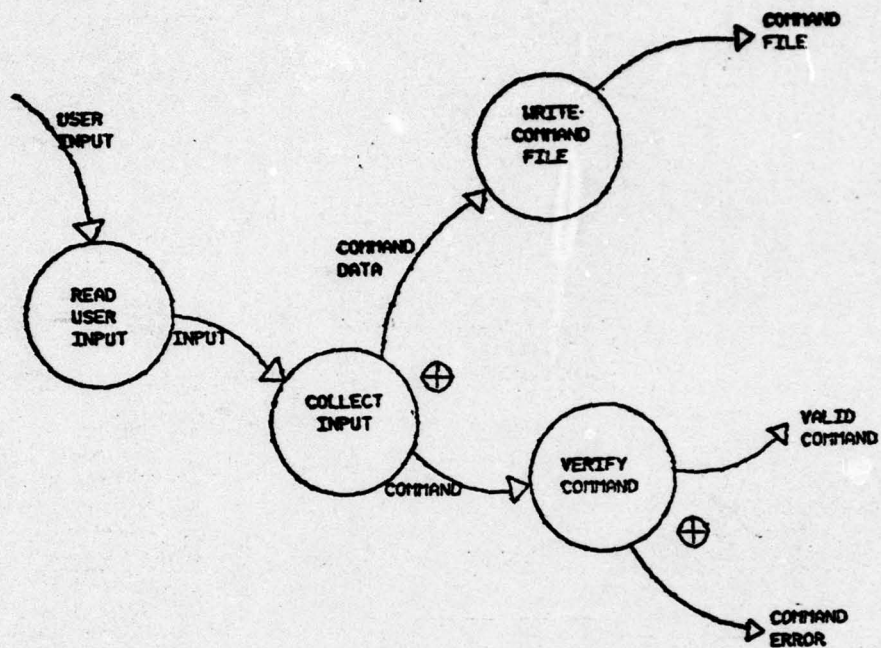
6.1   INTERPRET COMMAND.

TEXT:

Interpreting the command is identical to the function
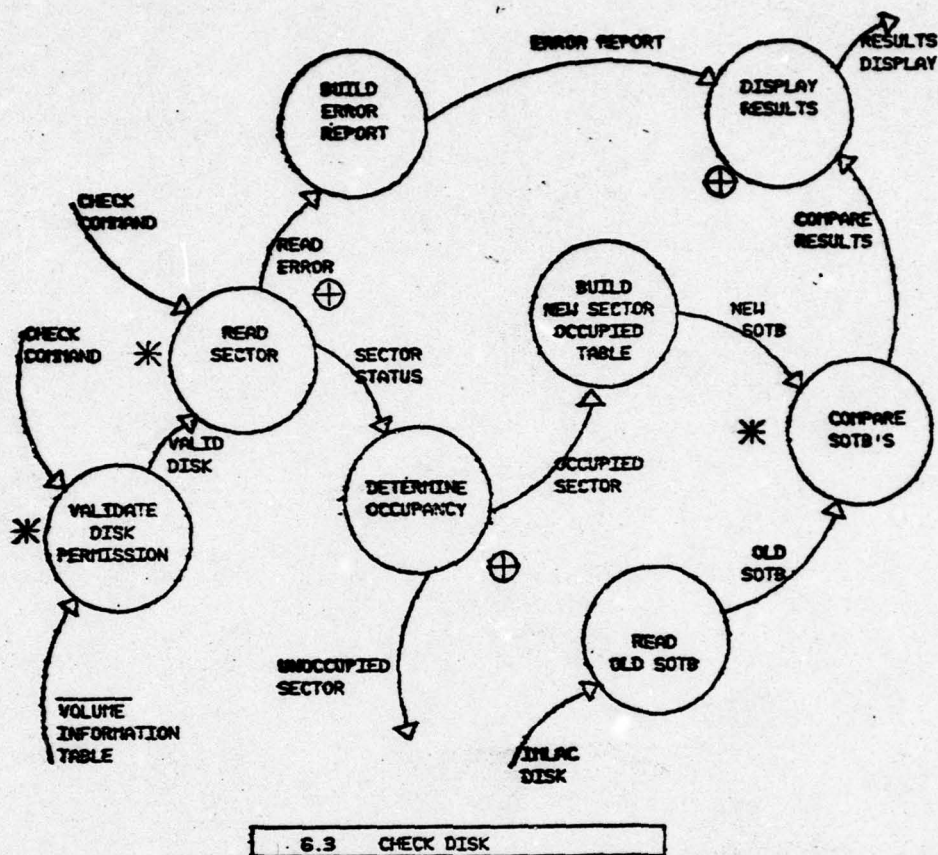described in diagram 2.1. It is included here for model
completeness.

6.2 COPY DISK

TEXT:

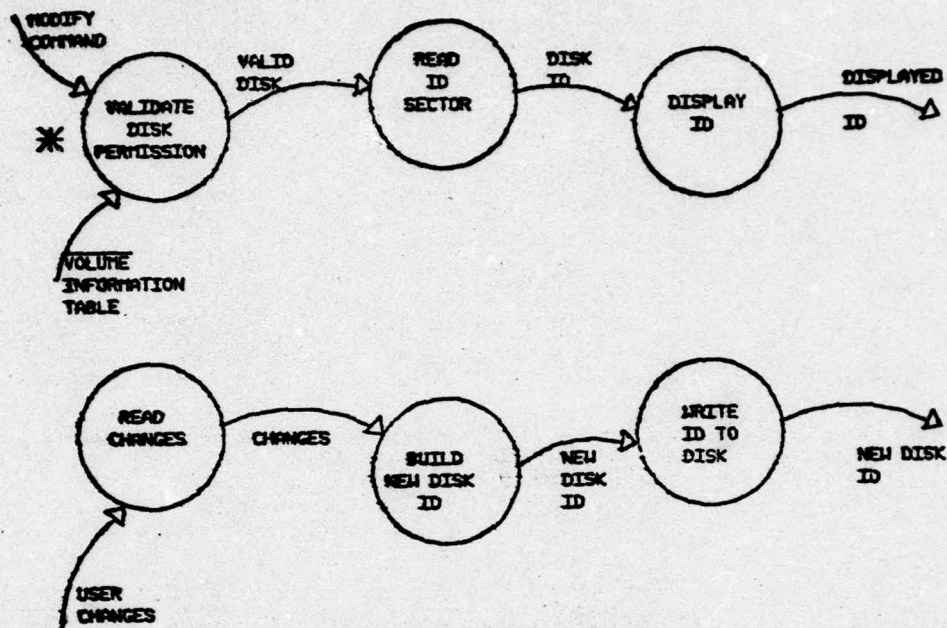The disk copy command contains the source and destination disk drive for the copy. The cartridges on these drives are validated for user permission to modify. The disk is copied a sector at a time with errors displayed.

6.3   CHECK DISK

TEXT:

Disk checking is a series of sector reads to determine if any error statuses occur. Another check is that the amount of occupied space on the disk agrees with the sector occupied table contained in a file on the disk. Errors from this comparison or from the read are reported to the user.
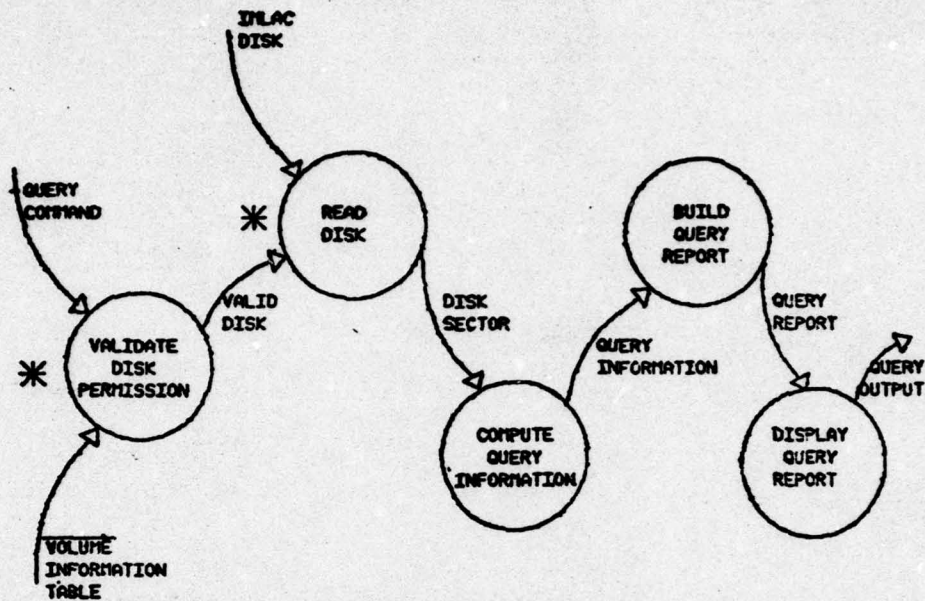
77

```
          MODIFY
          COMMAND
                                  VALID         READ        DISK         DISPLAY       DISPLAYED
                                  DISK          ID          ID                         ID
              VALIDATE                          SECTOR                    DISPLAY
        *     DISK                                                        ID
              PERMISSION


          VOLUME
          INFORMATION
          TABLE



              READ                 CHANGES                   NEW          WRITE          NEW DISK
              CHANGES                          BUILD         DISK         ID TO          ID
                                               NEW DISK      ID           DISK
                                               ID


          USER
          CHANGES
```

8.4	MODIFY DISK

TEXT:

    Modifying a disk ID is accomplished in two steps.
First, the user's permission is validated and the current
ID is displayed. Next, the user enters any changes to the
current ID which are used to format the new ID and written
to the disk.

```
                      INLAC
                      DISK

      QUERY                  READ            BUILD
      COMMAND          *     DISK            QUERY
                                             REPORT
                                                      QUERY
                             DISK                     REPORT
             VALID           SECTOR                          QUERY
             DISK                                             OUTPUT
    * VALIDATE                                       DISPLAY
      DISK                         COMPUTE           QUERY
      PERMISSION                   QUERY    QUERY    REPORT
                                   INFORMATION INFORMATION

      VOLUME
      INFORMATION
      TABLE
```

6.5	QUERY DISK

TEXT:

Disk queries consist of statistical information about the disk status. This information is computed by sequentially reading the entire disk and building the corresponding report.
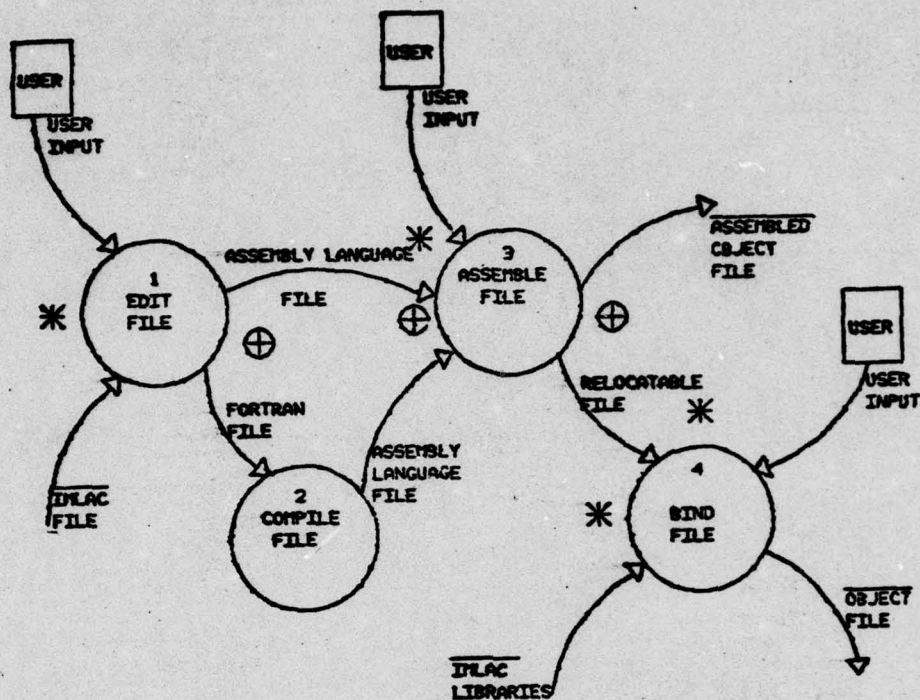
```
7.    PROGRAM CREATION
```

TEXT:

Program creation refers to the generation of executable object code from either a high order language or assembly language. The four steps in accomplishing this task are: creating the source file, compiling the source if it's a high order language, assembling the assembly code, and linking the file to the proper routines.

80

```
                                                    DYLAC
                                                    FILE

         USER                  CREATE                  STORE
         INPUT                  FILE                    FILE

              CREATE
              CMND                        CREATED
                                          FILE
         INTERPRET   ⊕                        ✳
         COMMAND         MODIFY                        MODIFIED
                         COMMAND      MODIFY            FILE
                     ⊕                FILE    ⊕

              LOAD
              CMND

              ✳         LOAD        LOADED
                        FILE        FILE
         DYLAC
         FILE
```
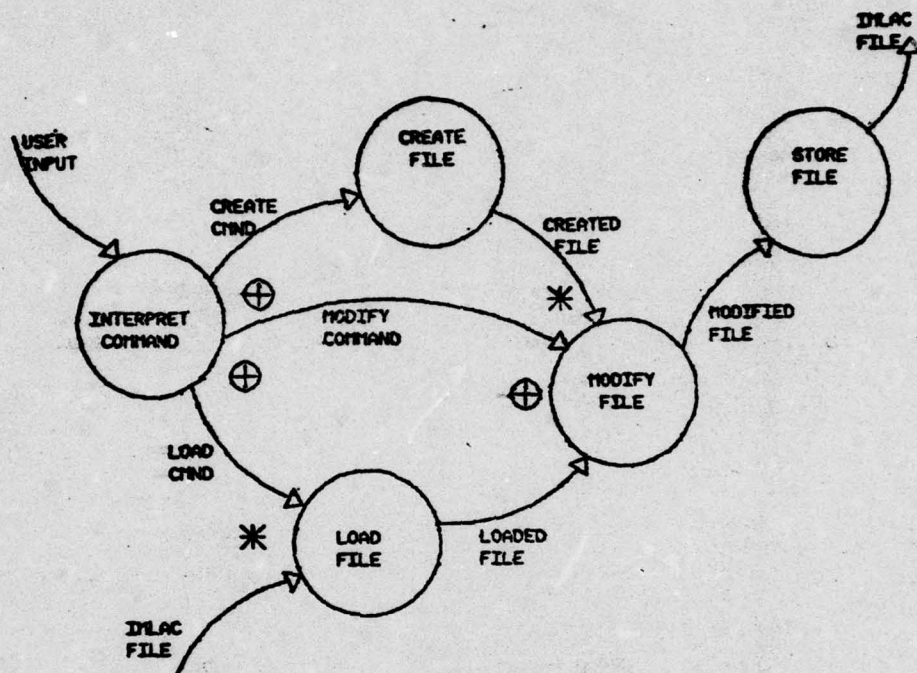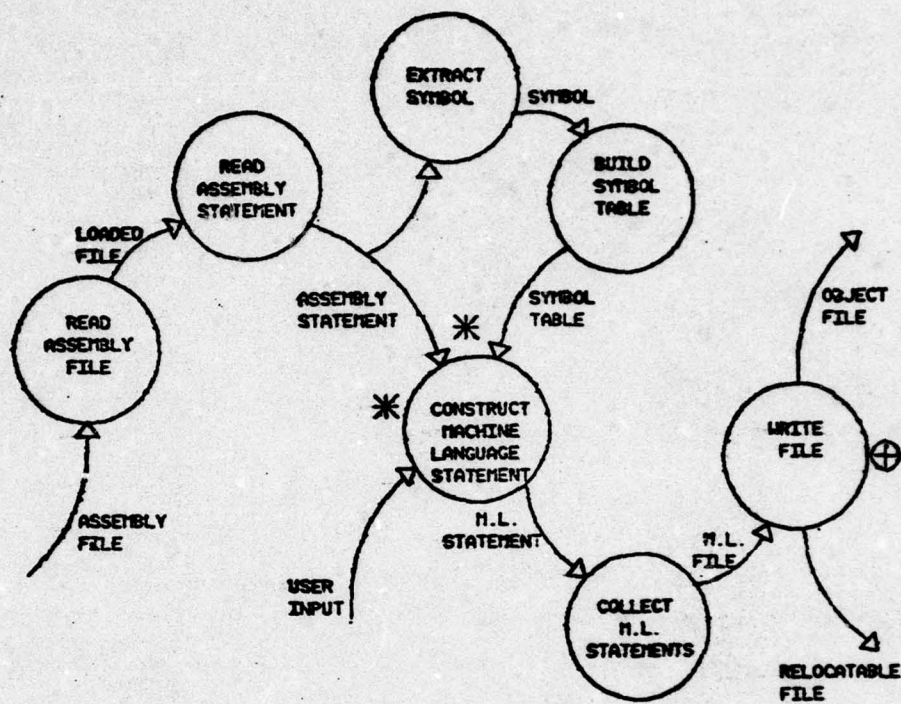
7.1	EDIT FILE

TEXT:

Editing a program file creates a new file if it did not exist before, or modifies an existing file. The user command specifies whether to load a file into memory for editing, add to the file in memory, or modify the file. The final version of the file is stored to the disk.

7.2 COMPILE FILE
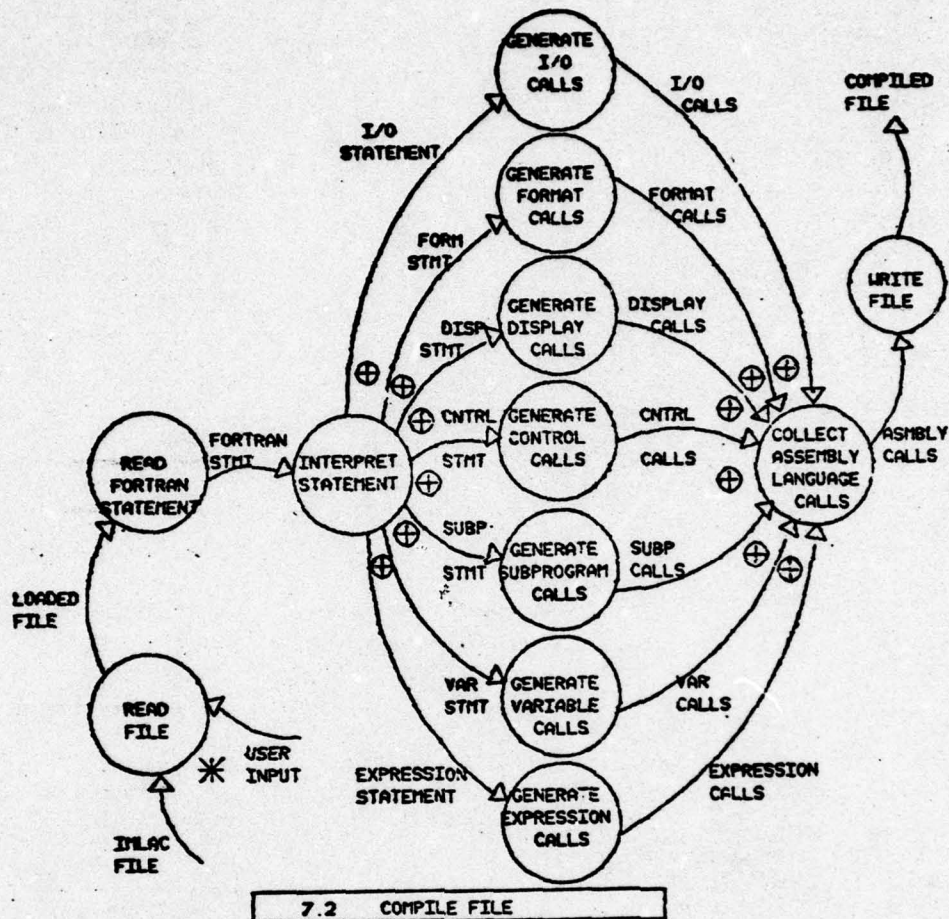
TEXT:

Fortran statements are separated into the seven types
shown in the diagram for compilation purposes. Each type
generates the appropriate assembly code to perform the state-
ment. This code is collected into one assembly file and
stored to the disk.

EXTRACT SYMBOL → SYMBOL

READ ASSEMBLY STATEMENT

BUILD SYMBOL TABLE

LOADED FILE

READ ASSEMBLY FILE

ASSEMBLY STATEMENT

SYMBOL TABLE

ASSEMBLY FILE

CONSTRUCT MACHINE LANGUAGE STATEMENT

OBJECT FILE

WRITE FILE

USER INPUT

M.L. STATEMENT

M.L. FILE

COLLECT M.L. STATEMENTS

RELOCATABLE FILE

| 7.3 | ASSEMBLE FILE |

TEXT:

The assembly process is done in three passes. Each statement is read for correct syntax and symbols to build the symbol table. The machine code is generated in either an object or relocatable format depending on the user specifications. Object code is complete and ready for execution while relocatable code must be linked with separate routines to satisfy external references.

83

```
USER
INPUT
       READ              UPDATE                           STORE
       USER              SYMBOL                           OBJECT
       INPUT             TABLE                            FILE
                                         UPDATED
                                         SYMBOL
                                         TABLE
            INPUT      RELOCATABLE                  OBJECT
                       FILE                         FILE
      *            READ                    BUILD
                   RELOCATABLE             OBJECT
                   FILE                    FILE              IMLAC
                                                            OBJECT
                                                            FILE
   IMLAC
   RELOCATABLE      ⊕
   FILE

              IMLAC
              LIBRARIES
```
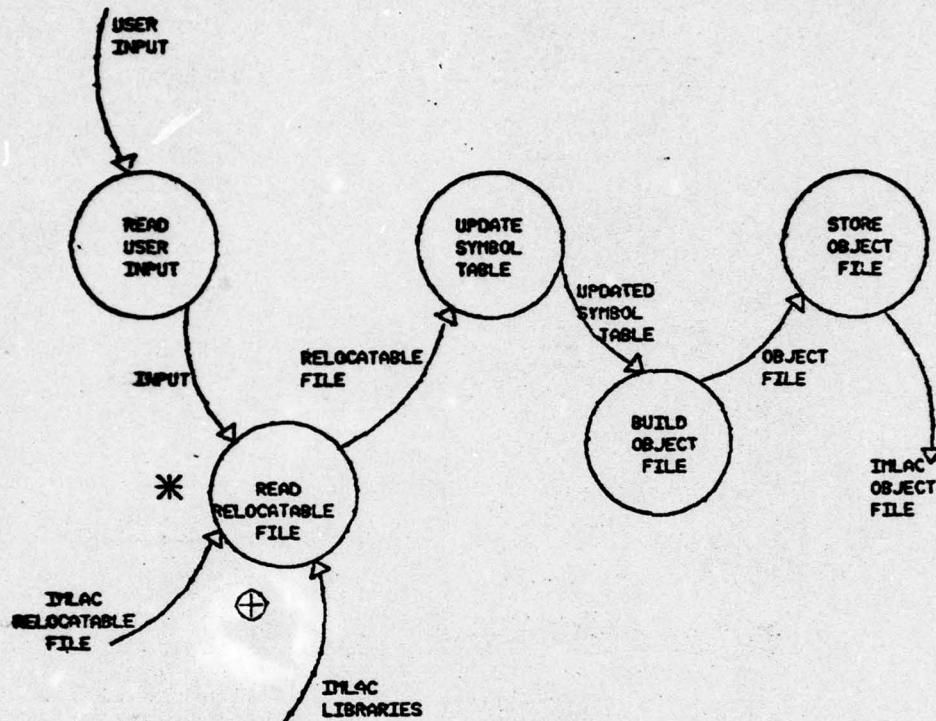
| 7.4   BIND FILE |

The binding process is for linking separate routines to the main machine language program to create executable object code. The user specifies the name of the routines to be linked together, and the type of linking required. The symbol table for each routine is updated to reflect the correct addresses, and the object code is generated using this updated table. The final object code is stored for later use.

84

Appendix B

Required System Model

DESIRED GRAPHICS SYSTEM
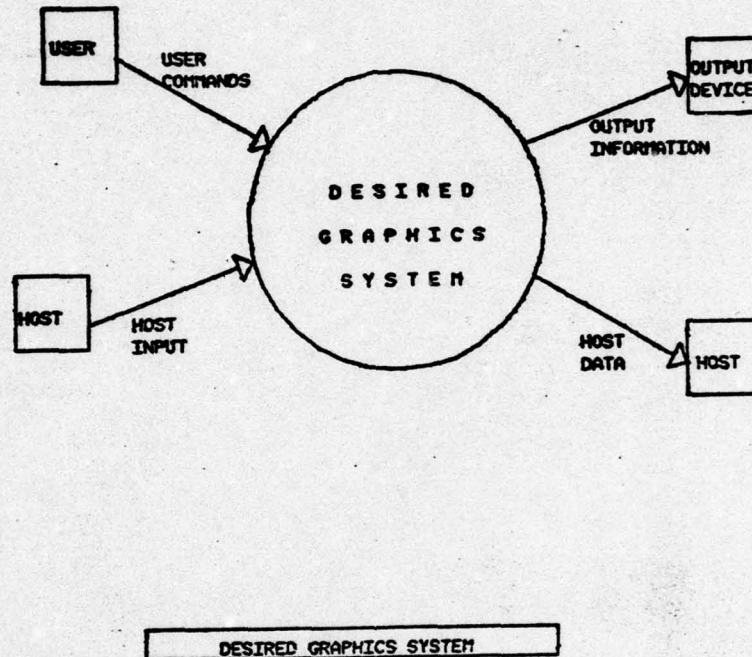
TEXT:

The desired graphics model represents the user's view-
point of the requirements for a graphics system. These re-
quirements are based on the particular application of the user.
The input and output to the system are the same as for the
existing system; host and terminal user inputs, and host and
peripheral outputs. The remainder of the system is modeled
following the same functional division as the existing system.
The disk maintenance function was omitted since no difference
was specified from the existing model. Within each function,
activities which are identical to the existing activity are
not decomposed to the second level. These will be specified
in the associated function text.

85

```
USER
COMMAND

        1
    INTERPRET
    COMMAND

    FILE
    NAME    *       2
                  LOAD
                  FILE        LOADED
                              FILE

    FILE
    DATA

    SYSTEM
    FILE

                                EXECUTING
                                PROGRAM

                        3
                    EXECUTE
                    FILE
```
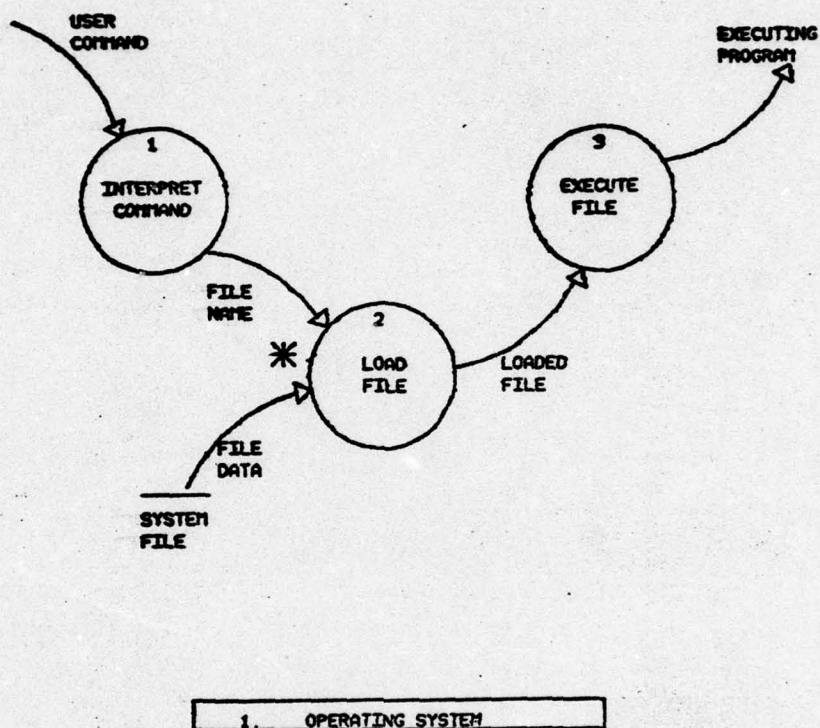
1.	OPERATING SYSTEM

TEXT:

The user command to the operating system is interpreted as a file name to be executed. The file is loaded from the disk and placed into execution. Activities 1.2 and 1.3 are the same as modeled in the existing system and are not expanded here.

86

INFORMATION FILE

SYSTEM INFORMATION

DISPLAY INFORMATION

DIS-PLAY

LOCAL MSG

DISPLAYED INFORMATION

USER COMMAND

BUILD LOCAL MESSAGE

MESSAGE COMMAND

DISPLAYED COMMANDS

READ KEYBOARD

KEYBD INPUT

COLLECT INPUT

LOCAL FILE NAME

DISPLAY COMMANDS

HELP COMMAND

FORMAT COMMANDS
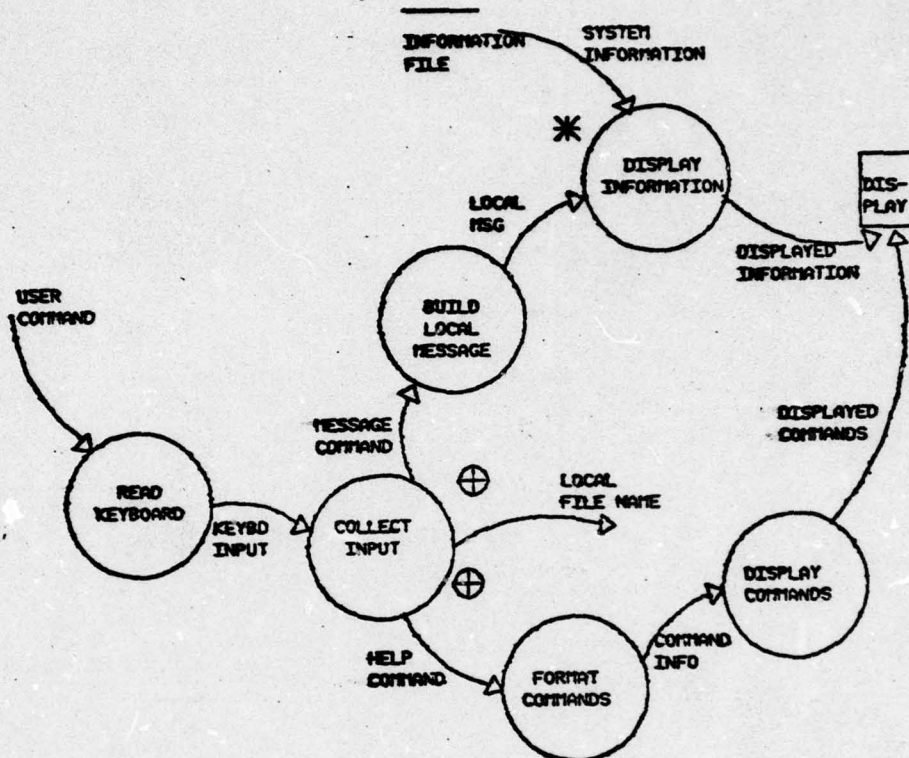
COMMAND INFO

| 1.1 | INTERPRET COMMAND |

TEXT:

The user input is interpreted as one of three types:
a file name to be loaded, a message command, or a help command.
The message command is used to specify a message to be dis-
played with the normal system information. This message remains
visible for all subsequent users, and provides a means to inform
system users of any critical information about the system. The
help command displays a summary of available system commands
eliminating the requirement for users to remember all command
formats. The file name input is the file to perform the desired
function.

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU

```
                                  DILAC
                                  FILE
   LOADED                                              2
   FILE                7                      *      LOAD              LOADED
                    RECOVER                                             FILE
                     FILE

                  RECOVER                                    3
                  COMMAND                          *       STORE         DILAC
   USER                                   STORE                          FILE
   INPUT              LOAD              COMMAND
                   COMMAND                       MASTER
                                                DIRECTORY        4
                      ⊕    ⊕                       *          DELETE
                                          DELETE                        NEW DISK
                            ⊕              CMND                         DIRECTORY
                     1
                  INTERPRET              MASTER
                  COMMAND       ⊕       DIRECTORY        5
                                                      MODIFY          MODIFIED
                                 MODIFY  *                            MASTER
                                COMMAND                               DIRECTORY
                      ⊕
                   QUERY
                  COMMAND                              6
                                                    QUERY            FILE
                                          *                        INFORMATION

                                 MASTER
                                DIRECTORY  ⊕  CHANNEL INFORMATION
```

2.	FILE MAINTENANCE
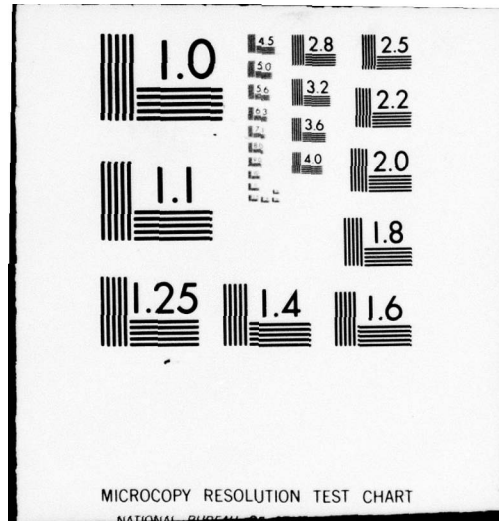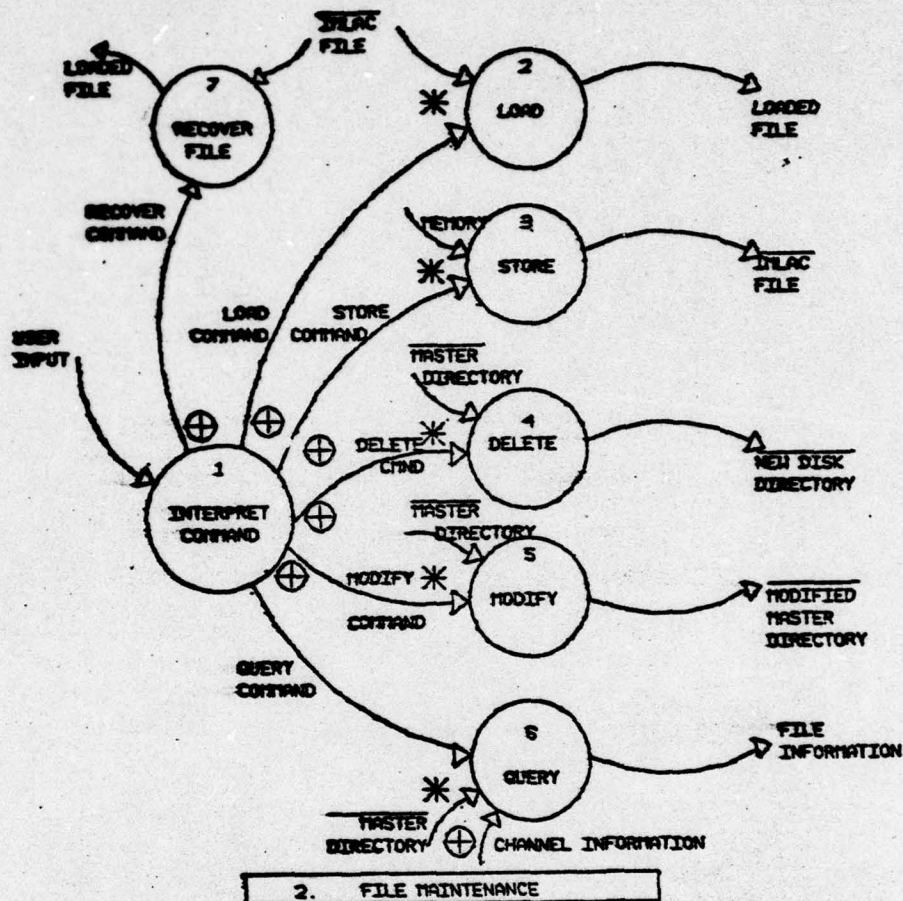
TEXT:

File commands are divided into the six specified func-
tions shown. The outputs of the different functions include
updated files, file information for user viewing, or files
loaded into memory. Activities 2.4 and 2.7 are the only
bubbles requiring decomposition. Descriptions of the re-
maining activities can be found in the existing system model.
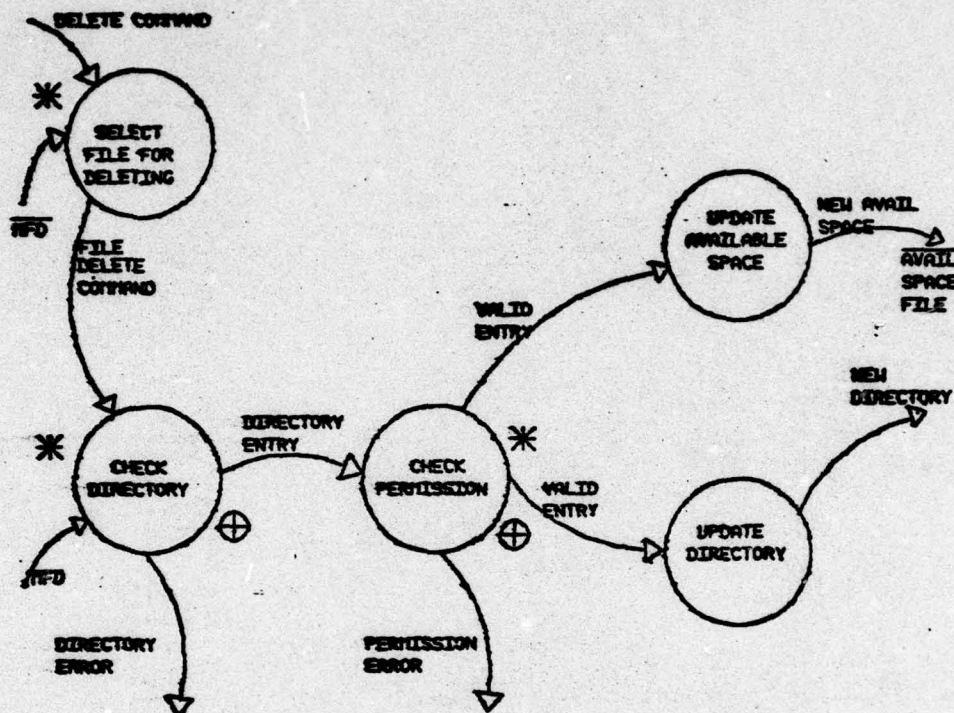
2.4   DELETE FILE

TEXT:

The first process in deleting a file is to specify which file(s) are affected. It is possible to delete more than one file providing for a general "clean-up" capability. Multiple file deletions require the specification of some key associated with each file in the desired group. The Master File Directory is searched for any files containing this key, and the file entry is removed if the permissions are valid. Both the available space table and directory are updated to reflect the change.

```
                    ┌────────┐
                    │ IMLAC  │
                    │ DISK   │
                    └────────┘
                         │
                       DISK
                      SECTOR
                         ↓
   RECOVER
   COMMAND              *    ┌──────────┐      FILE
      │                      │   READ   │─────SECTOR────┐
      │                      │  SECTOR  │               │
      ↓                      └──────────┘      ⊕        │
 ┌──────────┐    DISK           │                  ┌──────────┐    LOADED
 │CALCULATE │──ADDRESS──────────┘                  │   LOAD   │    FILE
 │   DISK   │                   READ        ⊕      │  SECTOR  │────────→
 │ ADDRESS  │                   ERROR              │   INTO   │
 └──────────┘                     │                │  MEMORY  │
   *                              ↓                └──────────┘
      │                      ┌──────────┐              │
     RFD                     │  FORMAT  │  FORMATTED    │
                             │  ERROR   │──MARKER───────┘
                             │  MARKER  │
                             └──────────┘
```
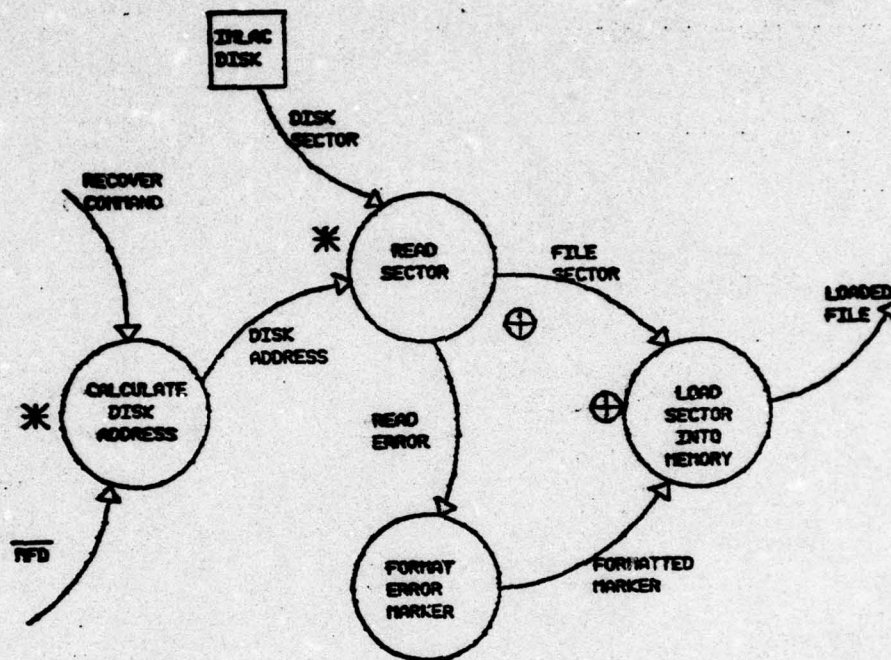
┌─────────────────────────────────┐
│ 2.7   RECOVER FILE   │
└─────────────────────────────────┘

TEXT:

Recovering a file refers to loading into memory all
parts of a file that can be accessed. Those parts unaccessable
due to hardware errors or incorrect file pointers are flagged
with error markers for user handling. Once loaded, the user
can restore the file and resave it onto the disk.
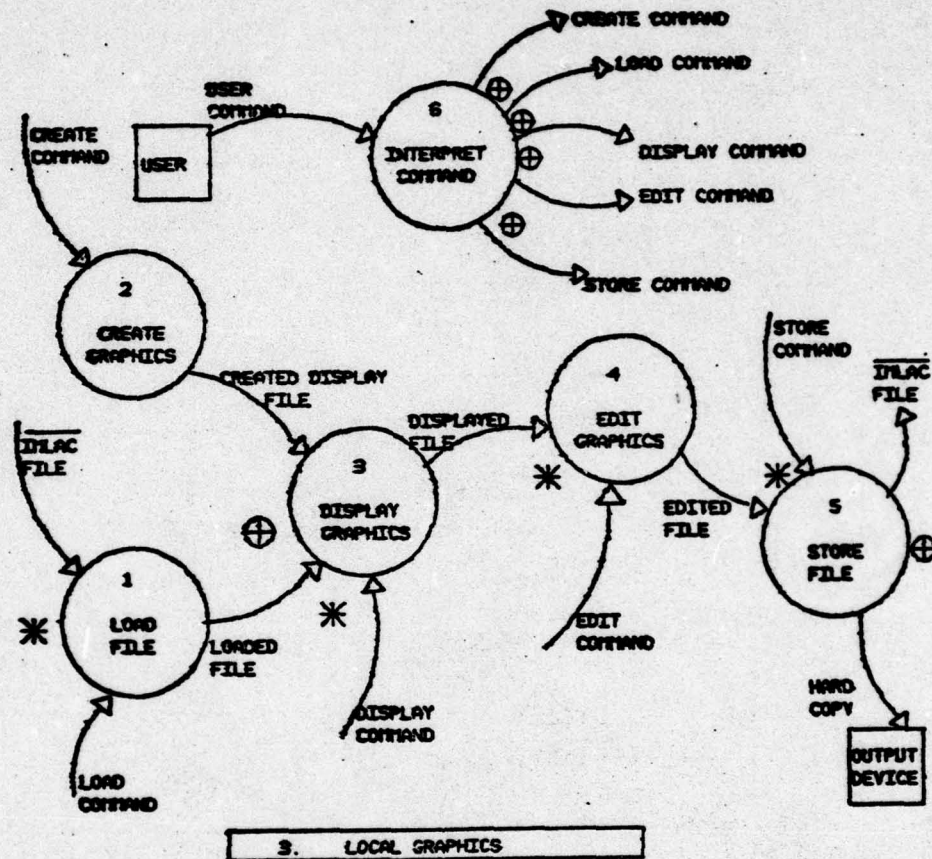
3. LOCAL GRAPHICS

TEXT:

Local graphics allow a terminal user to create, edit, and store graphic displays. Activities 3.3 and 3.5 are not decomposed to a second level. These activities can be located in the IMLAC system model.
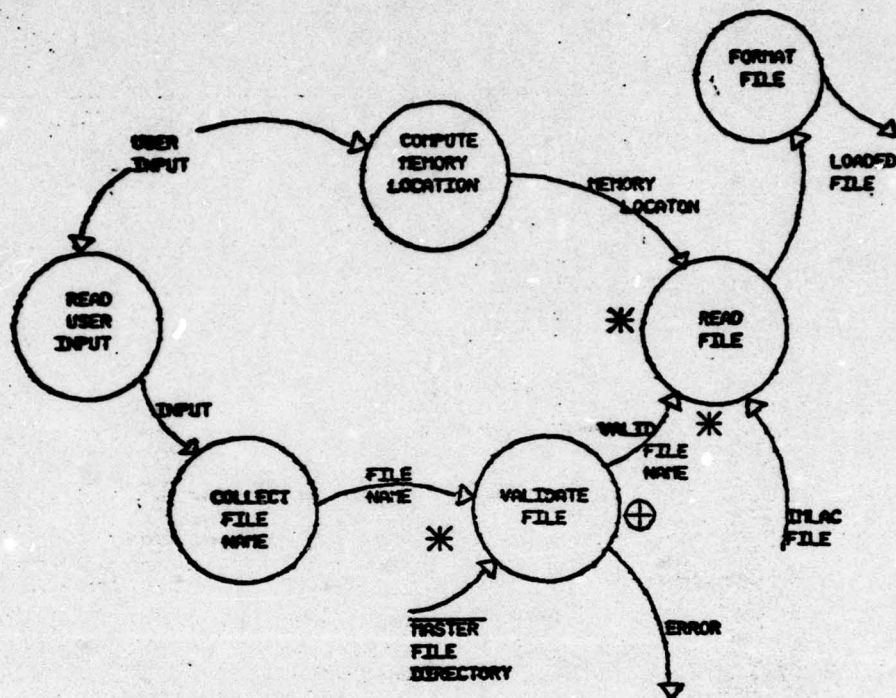
3.1   LOAD GRAPHIC FILE

TEXT:

     User input specifies the file to be loaded and the
location in memory for loading. There is no single format
required by the file to be loaded. Assuming it contains
graphic instructions in some format, it can be altered to
the necessary structure for local processing. This enables
graphic files of several different origins to be accessed
and modified in a local mode.
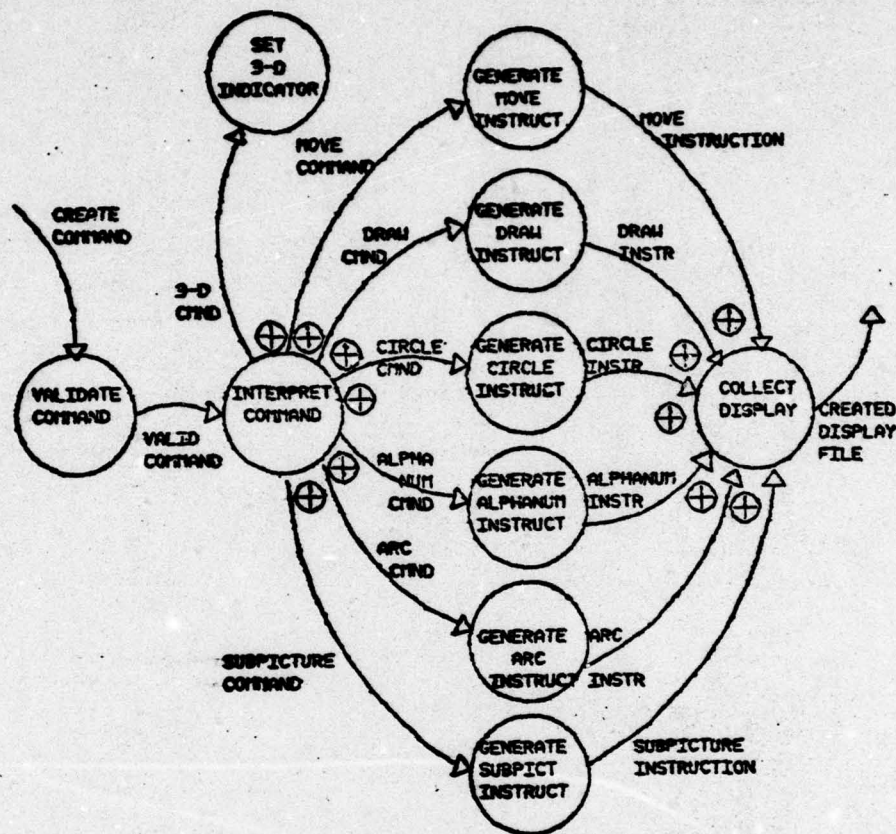
3.2 CREATE GRAPHICS FILE

TEXT:

The user can specify several different types of in-
structions to create graphics. The 3-D indicator allows
the user to create the display in three coordinates. Each
instruction generator uses this indicator to create the
proper commands.

93
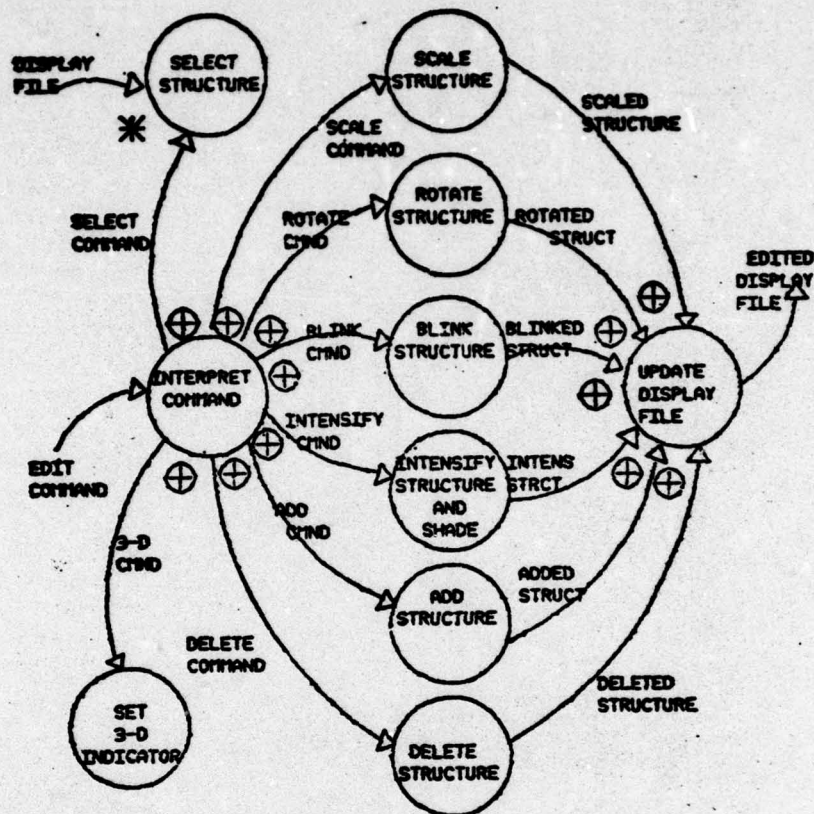
3.4 EDIT GRAPHICS FILE

TEXT:

To edit the graphic display, the user must indicate
which item on the screen is to be modified, and whether the
modification is to be two or three-dimensional. Several
different modifications are available including a shading
or cross-hatching capability. This capability will fill in
a closed structure with the desired design.

94
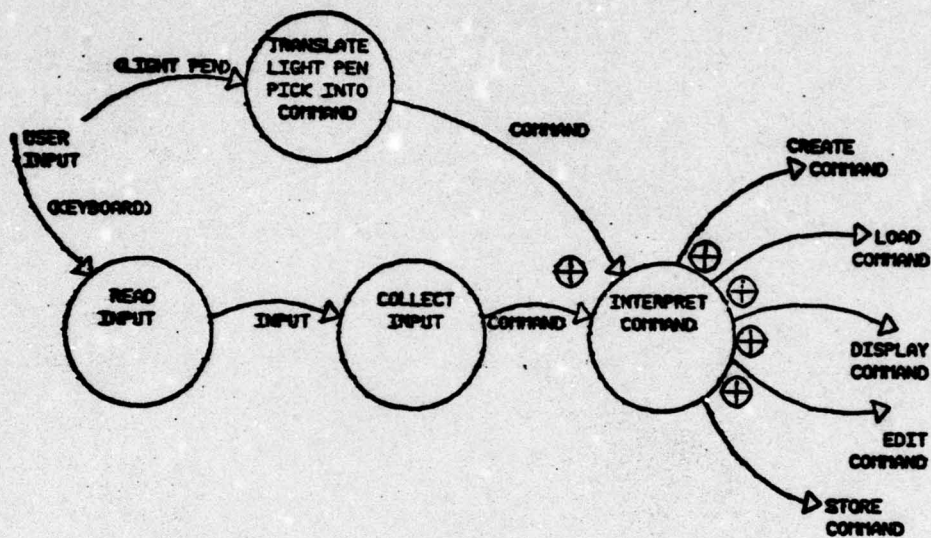
```
┌─────────────────────────────────┐
│  9.6    INTERPRET COMMAND        │
└─────────────────────────────────┘
```

TEXT:

     User commands can be inputted by the keyboard or the use of a light pen menu. The menu lists the various commands that are available for the current mode of operation, and the user chooses the desired item through a light pen pick. The command is then interpreted to the appropriate function.

```
         6                        5              ⊕──▷ CREATE GRAPHICS
     GENERATE        HOST      INTERPRET
       HOST        COMMAND      COMMAND        ⊕──▷ QUERY GRAPHICS
     COMMAND                                   ⊕──▷ EDIT GRAPHICS
   FORTRAN STATEMENT                                ──▷ DISPLAY GRAPHICS

   CREATE
   COMMAND                    QUERY
                             COMMAND
                                              3
              1                            QUERY              ──▷ HOST
           CREATE                         GRAPHICS              INFORMATION
          GRAPHICS                   *

   CREATED              DISPLAY
   DISPLAY              FILE
   FILE                                          4               EDITED
              2                                EDIT              DISPLAY
       *   DISPLAY                       *    GRAPHICS      ──▷
           GRAPHICS

   DISPLAY                  EDIT
   COMMAND                 COMMAND
```
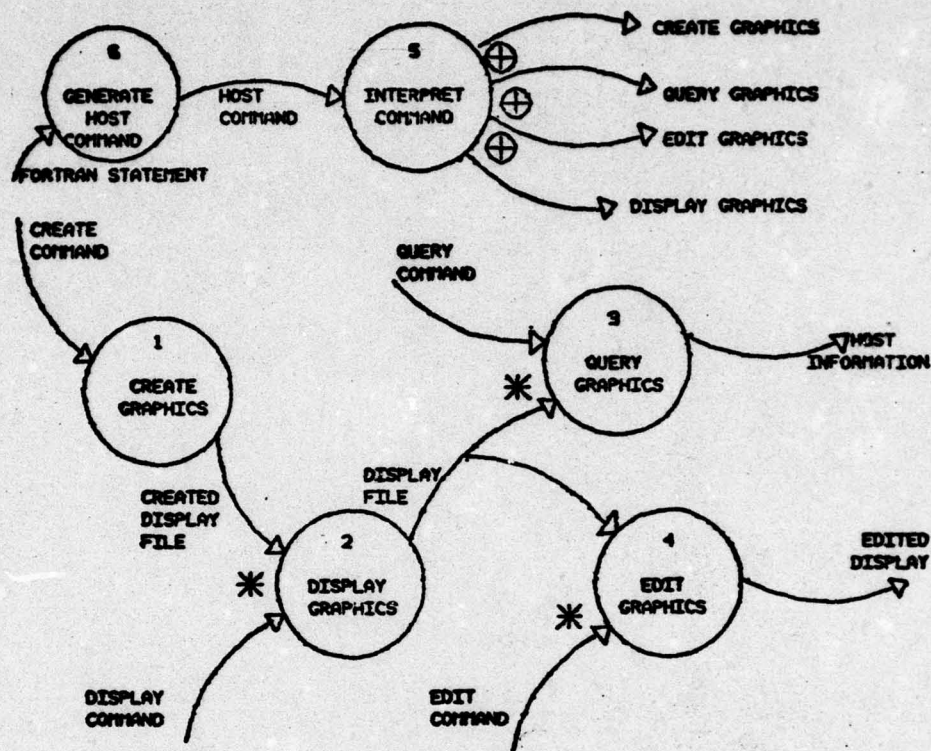
4.	HOST GRAPHICS

TEXT:

Host graphics extend the boundaries of the system to include the generation of the commands on the host. These commands are transmitted to the system, and processed accordingly. Information can be transferred from the local system by the user through the query function. Activities 4.2, 4.3, and 4.5 are not further decomposed. Their functions can be viewed in the existing system model.

96

4.1 HOST CREATE GRAPHICS
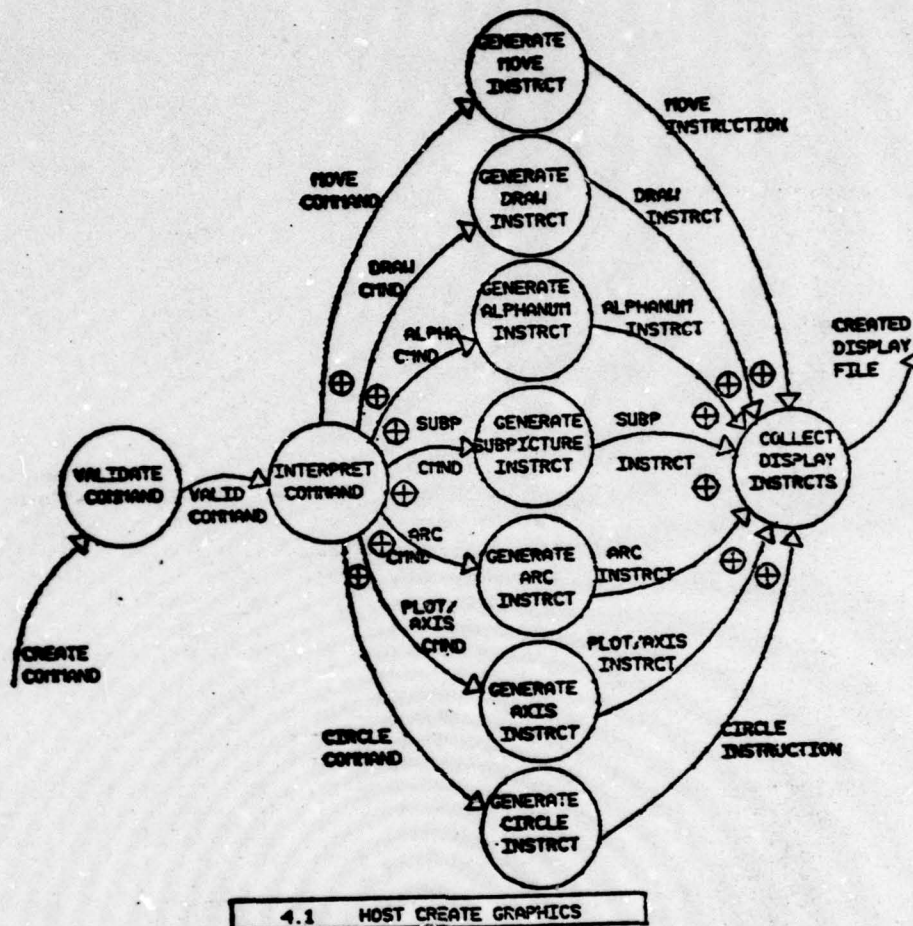
TEXT:

Several instructions are available which are similar to the local graphics creation. The additional command creates an x,y axis and plots a buffer of points for the user. The generated instructions are displayed on the screen.

97

DISPLAY FILE → SELECT STRUCTURE

SCALE STRUCTURE

SCALED STRUCTURE

SCALE COMMAND

SELECT COMMAND

ROTATE CMND → ROTATE STRUCTURE — ROTATED STRUCT

EDITED DISPLAY FILE

BLINK CMND → BLINK STRUCTURE — BLINKED STRUCT

INTERPRET COMMAND

UPDATE DISPLAY FILE

INTENSIFY CMND → INTENSIFY STRUCTURE — INTENS STRCT

EDIT COMMAND

ADD CMND → ADD STRUCTURE — ADDED STRUCT

3-D CMND

DELETE COMMAND

DELETED STRUCTURE

SET 3-D INDICATOR

DELETE STRUCTURE

4.4   EDIT GRAPHICS FILE

TEXT:

     Host graphic editing is identical to local editing.
The only difference is the origin of the commands. In local
graphics, the commands are issued by the user at the terminal,
while in host graphics, they are generated by the host com-
puter.

```
FORTRAN
PROGRAM
STATEMENT

                                    FORTRAN
              FORTRAN                PARAMETER
              ROUTINE
  SEARCH                  EXECUTE                GENERATE            HOST
  FORTRAN                 FORTRAN                GRAPHICS            COMMAND
  LIBRARY                 ROUTINE                COMMAND
```
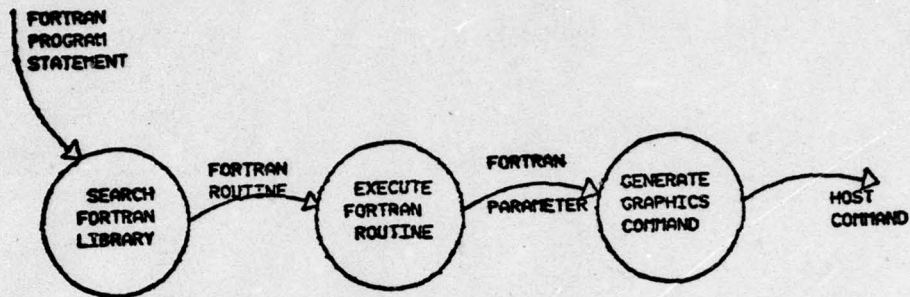
```
4.8    GENERATE HOST COMMAND
```

TEXT:

    Host commands are generated from an executing program
on the host computer. The program contains calls to library
routines which form the necessary graphic commands for the
local system and transmits them to the system.

99

```
┌──────┐
│ HOST │
└──┬───┘
   │ HOST
   │ INPUT
```

      1   4   ┌─────┐
   DEFINE   WRITE   │INLAC│
  TRANSMIT   INLAC   │FILE │
 PARAMETERS   FILE   └─────┘

         XMIT   2   FILE
        PARAMS   INTERPRET   INPUT
                     HOST
  USER   INPUT   DISPLAY   5   DISPLAY DATA
 COMMAND   INPUT   OUTPUT   ┌────────┐
┌──────┐   DISPLAY   │DISPLAY │
│ USER │   └────────┘
└──────┘
           FILE   INLAC FILE
          COMMAND
                                            6   HOST   ┌──────┐
     3   RECEIVE   SEND   DATA   │ HOST │
 INTERPRET   FILE CMND   INLAC   └──────┘
   USER   XMIT   FILE
   INPUT   PARAMS

                    NORMAL   7
                    INPUT   SEND   ┌──────┐
  USER   INPUT TO   │ HOST │
  INPUT   HOST   └──────┘
┌──────┐   HOST
│ USER │   XMIT   DATA
└──────┘   PARAMS

                 ┌───────────────────────────┐
                 │ 5.   TTY MODE   │
                 └───────────────────────────┘

TEXT:

       Communication to a host time sharing system is handled
by the TTY function. The user defines the appropriate par-
ameters for the transmission, and proceeds to send and receive
data with the host machine. The user can specify entire files
to be received or transmitted between the local and host
system. Activities 5.3 and 5.7 are the only bubbles expanded.
The remaining bubbles are identical to the existing system
model.
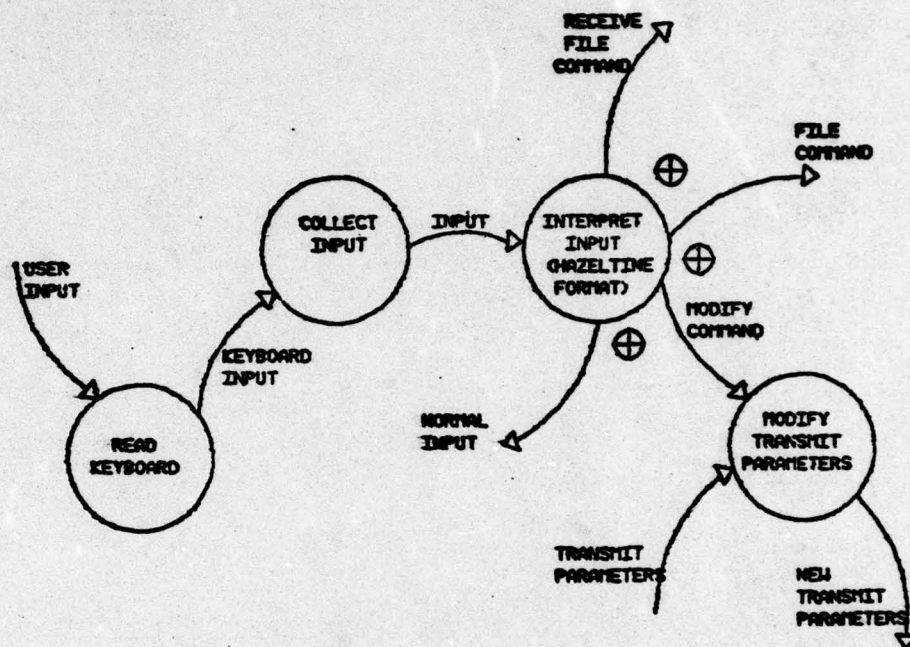
5.3   INTERPRET USER INPUT

TEXT:

   User input is interpreted in a format like the Hazetine
terminal. This format requires users to only be familiar
with one set of TTY commands to communicate with either ter-
minal. The input is interpreted to modify parameters, send
to the host, or specify file transmission.
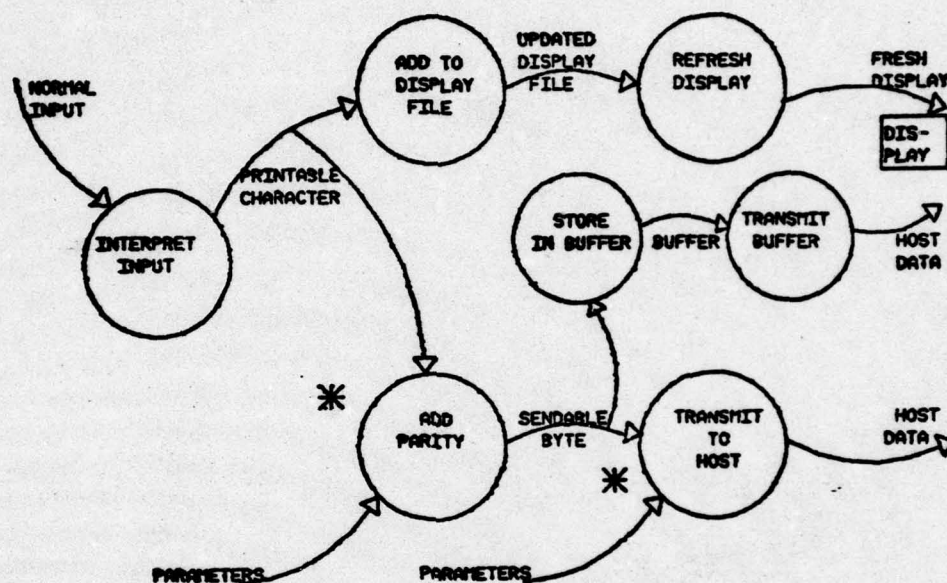
5.7 SEND INPUT TO HOST

TEXT:

Information destined for the host is displayed on the screen and formatted for transmission. The specified parameters define whether the information is sent directly or stored in a local buffer. The entire buffer is transmitted at one time upon user request.
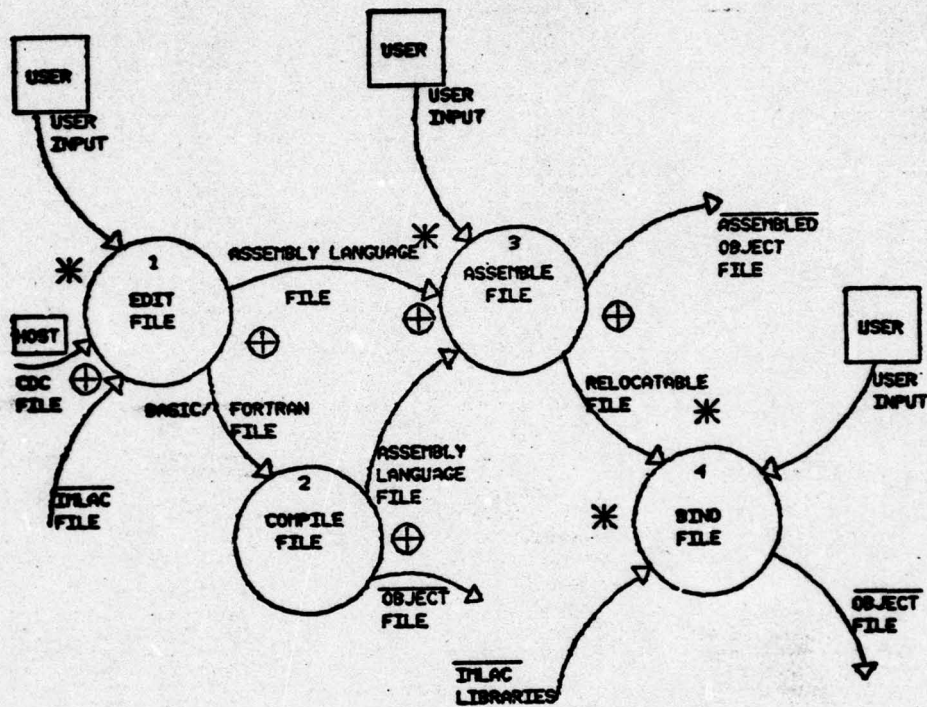
7.   PROGRAM CREATION

TEXT:

    Object code for execution is generated in four steps:
first creation, then compilation, assembly into machine
language, and finally the linking of external routines.
This sequence can be altered at any step depending on user
specification. For example, either a high order or assembly
language porgram can be edited, compilation can create as-
sembly or object code, and the assembly can produce relocatable
or object code. Only bubble 7.2 is decomposed.

103

7.2 COMPILE FILE.
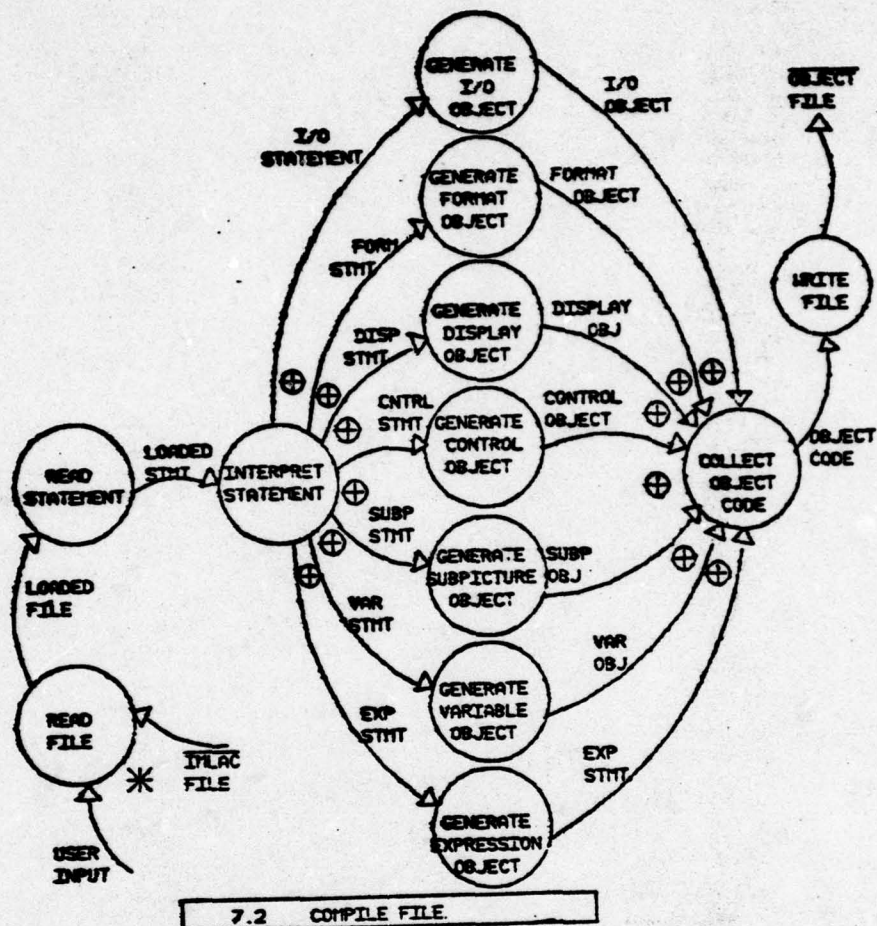
TEXT:

The IMLAC system model illustrates the compilation of a file into assembly language. This diagram shows the compilation directly into object code. The user specifies which format is to be generated depending on the particular application.

104

Appendix C

IMLAC Model Comparison Results

The results of the structured evaluation performed on the IMLAC PDS-4 graphics system are presented in two parts: discrepancies and recommendations. The discrepancies summarize by function the differences found during the model comparison phase described in Chapter III. Recommendations based on this comparison were presented to the FBR organization for consideration. These suggestions represent possible solutions to the discrepancies.

Model Discrepancies

Each function in the two IMLAC models were compared for differences which represent deficiencies in the satisfaction of user requirements. These differences are presented by function with a brief description of the associated requirement as specified by the user.

Operating System

The major discrepancy found in the operating system function is the addition of two commands to the MONITOR. These commands, MESSAGE and HELP, are required to encourage the utilization of the IMLAC system by simplifying system operation. MESSAGE allows the system manager to display helpful information or notices to all users of the system. HELP provides online command descriptions to permit an inexperienced user to look up individual command formats without requiring a search through system manuals.

File Maintenance

A file recovery option is the only discrepancy illustrated in this function. When a file is partially destroyed through hardware failure or user error, it is required that some means be available to recover the portion of the file still intact. The destroyed part of the file should be marked for user awareness to permit the rebuilding of the appropriate sections.

Local Graphics

Several discrepancies exist in the local graphic function. The ability to format files is one difference in the "Load Graphic File" process. This capability represents a requirement to use a graphic file from any originating source and be able to perform local editing and displaying of that file. The current system requires a strict format to perform editing, and limits the type of files that can be loaded for local manipulation.

A three dimensional capability is a discrepancy in both the creation and editing of local graphics. The type of graphic work performed by the users require the ability to view and modify images in three dimensions.

Light pen command specification is a requirement to simplify user interaction for local image processing. This capability eliminates complex key combinations and requires less knowledge about operating instructions.

Host Graphics

A major difference in the host graphic function is the generation of graphic commands on the host computer. Command generation is required to eliminate complex programming in the host program to transmit detailed command specifications. The availability of automatic command generation permits more general Fortran graphic calls.

The ability to generate x,y axis and automatic point plotting is a difference in the "Host Create Graphics" process. The need to analyze data results quickly in a graphic plot

107

is the requirement for this addition.

Current editing capability is insufficient to satisfy the user requirement for interactive image modification. The required system illustrates full editing capability similar to the local graphics commands.

TTY Mode

The requirement to simplify user interaction is the basis for the Hazeltine command discrepancy. By allowing the user to use the same commands on both terminals, the requirement for learning a new set of commands is eliminated.

The "Send Input to Host" process illustrates a discrepancy in sending a data buffer versus a data byte. This capacity allows for extensive screen editing prior to transfer to the host machine, and represents greater flexibility in host communication.

Program Creation

Discrepancies in the program creation function deal mainly with the compiler process. To increase speed and simplify user interaction, the requirement for direct compilation to object code is generated. To accomodate users familiar with BASIC, the requirement is specified for a BASIC compiler.

IMLAC Recommendations

The following set of recommendations is presented to the Analysis and Optimization Branch of the Air Force Flight Dynamics Laboratory, WPAFB, for consideration toward improving the IMLAC PDS-4 graphics system. These recommendations are a result of an evaluation completed in partial fulfillment of the requirements for the degree of Master of Science at the Air Force Institute of Technology.

The recommendations are separated into three categories: hardware, software, and procedural. A brief descriptive text is included with each recommendation. The recommendations are ordered in each category according to the relative priority within the group.

Hardware

1. Acquisition of a typewriter quality printer. To facilitate the use of the flexible text editor, it is necessary that a high quality print output be available. This device would encourage the use of the system for documentation and report generation. Several devices compatable with the IMLAC system are currently available commercially.

2. Investigation of IMLAC improved Monitor CRT. A large amount of graphic information being displayed results in a detrimental "flickering" of the screen image. This problem is caused by the time delay required to refresh the screen images before the phosphorous begins to fade. IMLAC

has a CRT available which reduces this problem by increasing the length of time that the screen phosphorous remains active. A disadvantage of this device is that it will tend to produce "ghost" images during animation. Further analysis should be performed to determine if the IMLAC device is cost effective for FBR application.

3. Investigation of local host minicomputer. The acquisition of a local minicomputer would satisfy several graphic requirements not currently satisfied with the IMLAC system such as:

- greater local computing power
- dedicated host graphics
- real-time interactive graphics
- improved mathmatical accuracy
- standard Fortran compiler

However, the cost effectiveness of such a system is dependent on the amount and type of graphic work load projected by the FBR organization. Studies should be conducted to determine future requirements in the area of interactive graphics. Existing conditions indicate that any acquisition will not be effective unless substantial software and personnel support is obtained with the minicomputer. Examples of additional support required are:

- in-house software expertise (operating system, graphics, assembly language, etc.)
- interfacing graphics software for the IMLAC
- complete high level language capability

4. Addition of memory. Current requirements and utilization of the IMLAC system does not justify the acquisition of the IMLAC expanded memory option. Memory size is not a

limitation in current applications. Expanding the utilization of the IMLAC system via addition of a local host minicomputer will not necessarily create the requirement for the extra memory. A reevaluation of this option will have to be accomplished at that time.

Software

1. Modification of Fortran compiler. To promote utilization of the local processing capabilities, it is necessary that several modifications be implemented in the Fortran compiler. Specific areas requiring investigation are:

- memory limitation on size of subprogram
- fatal stack overflow problem
- improved graphic calls (including light pen)
- more definitive error messages

Currently, the FBR organization does not have the technical resources to accomplish such modifications locally. The cost of contracting out this project should be investigated, or an effort should be directed at obtaining local software expertise.

2. Creation of a local Fortran library. Several utilities can be generated in a combination of Fortran and embedded assembly language code to provide for better local processing capability. These utilities can be placed in a local library available to all Fortran users. Possible routines include:

- improved graphic calls
    a) light pen calls
    b) graphic macros
    c) automatic axis and plotting
    d) rotation, translation, and scaling
- input/output routines
    a) standardized calls
    b) host machine communication

111

c) peripheral interfaces
- mathmatical algorithms

3. Modify TTY interface (LINDI). Communication
protocol should imitate Hazeltine procedures to provide
maximum utilization of the IMLAC TTY capabilities. (This
modification was completed in the course of this study.)

4. Development of Graphics Compatability System (GCS)
interface. A three dimensional version of GCS is currently
available on the CDC with an IMLAC driver. Problems asso-
ciated with the driver have made the system unusable, and
are currently under investigation by the computer center.
Close scrutiny of the computer center's progress should be
continued to develop the GCS capability. This library is
essential to the host graphic function of the IMLAC system.

5. Generation of operating system utilities. Several
PDS-4 utilities can be generated to assist in normal system
utilization. Some examples of desirable utilities are:
- message line in Monitor (completed)
- character size control for utilities (completed)
- online command summary
- file recovery to salvage damaged files
- format conversion for local graphics (TIS4 format)

6. Modification of TIS4 for print/plot. The recently
acquired capability of combined plotting and printer generated
characters on the Versatec printer provides an opportunity
to create local graphics with high quality characters. This
modification involves adding a new command to the TIS4 program.

7. Modification of local utility character sets. PDS-4 modules contain character descriptions for displaying alphanumeric information to the display screen. Improved quality of these characters when plotting to the printer can be obtained by redefining the strokes which generate the characters. The SYMBL4 program can be used to help define the new characters. (This effort has been started in the course of this study, and TIS4 already has been modified.)

8. Modification of text editor (JOD). The JOD system has many useful attributes for text editing, but must be modified to generate reports in a more general format. These modifications should include:
- specified location of page numbers
- selection of section numbering schemes
- deletion of IMLAC product statement

Some of these changes can be accomplished by redefining the editor macros used by JOD.

9. Generation of STAGING driver. STAGING is a graphics system specifically designed for structural application. This program is directly associated with the mission of the FBR branch and represents a prine tool in the organization's work effort. To incorporate the refresh capability of the IMLAC system, a STAGING driver is needed to interface with the IMLAC. This driver should be written by a programmer familiar with the IMLAC system.

Procedural

1. Development of general IMLAC user's guide. A

113

summary of available capability and introduction to the documentation is required as a reference for the IMLAC users. (This guide was completed in the course of this study.)

2. Documented procedures for system use. A formal set of operating procedures should be generated and enforced by the FBRC branch. These procedures should include:

- log in requirements
- proper use of removable disks
- access to individual user areas
- log out and shut down procedures

These procedures will protect users against accidental destruction of files and equipment.

3. Class presentation of IMLAC system. A formal seminar of IMLAC capabilities and limitations should be presented to interested users. This seminar should be given regularly for new users as an introduction to the system. The information will encourage proper use of the IMLAC system.

4. Acquisition of IMLAC technical expert. Several recommendations have been made concerning modifications of IMLAC software and creation of assembly code utilities. There is sufficient need for an IMLAC technical expert to warrant a full time worker in this position. Such an expert could provide technical advice to system users, generate helpful utilities and modifications, and provide general graphic consultation to the organization.

Summary

The recommendations presented are based on an analysis of the current IMLAC system and perceived requirements of the FBR organization. A change in the graphic requirements would necessitate a reevaluation of the suggested efforts.

A suggested approach to implementing these recommendations is to begin with the procedural category. These are the easiest to fulfill, and will produce the quickest results. The first three recommendations in this section are aimed at increasing user awareness concerning the capabilities and limitations of the IMLAC system. This awareness will result in increasing system utilization. The acquisition of a technical expert is essential to the success of the suggested software modifications. Whether this expertise is developed with in-house personnel or with a new position, it is critical that some person or group be delegated specific responsibility to learn the IMLAC system in detail.

The first four software recommendations are the most critical to resolve current problems. Immediate attention should be directed to generate a useable Fortran compiler and to improve the GCS interface. These utilities are the most essential components for satisfying current user requirements. If a lack of expertise prevents the possibility of compiler modification, alternative measures should be initiated to reduce the problem. For example, known problems and ways of avoiding them should be documented by those users most familiar with the Fortran problems. The remaining software recommendations represent more of a convenience than a

direct requirement.

None of the hardware recommendations are immediately required for effective use of the IMLAC system. The typewriter printer would provide the most immediate increase in system utilization. The local host computer could produce a great many advantages, but only if the software and personnel requirements are first satisfied. The needs for technical expertise and strong software support will increase upon acquisition of an additional computer.

Appendix D

IMLAC Functional User's Guide

Contents

The IMLAC PDS-4 graphics system was leased in April 1977 to provide local computer support for structural analysis problems. This user's guide is provided to aid in and encourage utilization of the IMLAC's resources.

The content of this document is a direct by-product of an AFIT thesis effort to evaluate the existing IMLAC system. The evaluation was performed by Capt. Dennis L. Schweitzer from August 1978 to March 1979. All opinions and suggestions contained herein reflect the author's personal viewpoint.

1.1 Purpose of Manual

This user's guide is designed as an introduction and reference to the functional capabilities of the IMLAC system. Each section represents one functional aspect of the system. An overview of each function is given, and the appropriate programs and supporting documentation are listed. Limitations and problem areas are described where appropriate.

1.2 Manual Format

Chapter 2 of this manual provides an overview of the IMLAC system, and is recommended reading for all users. A suggested set of operating procedures for normal system use is included. Additional introductory material can be found in the overview section of each chapter.

Chapters 3-8 describe functional capabilities:

120

To effectively use this manual as a reference guide,

the following procedures are recommended:

1. Find the particular functional area of interest in
   the Table of Contents.

2. Read the overview of the area to become familiar
   with the general philosophy of operation.

3. Review the short descriptions of supporting pro-
   grams available for the function.

4. Refer to the IMLAC manual of operation for the sup-
   porting program to be used. This will provide
   specific instructions and details of operation.

2.1 Physical Characteristics

The IMLAC PDS-4 is a 16-bit word minicomputer which includes the following supporting hardware:

- 32K (decimal) memory
- 21 inch refresh vector graphics CRT
- Dual hard disk mass storage (10 million bytes/disk capacity)
- Versatec printer/plotter
- Light pen
- Control console

Two processors are resident with two separate supporting instruction sets. The display processor interprets display instructions for controlling all screen activities. These include vector generation, scaling, rotation, reflection, and blinking. These instructions reside in central memory and are accessed via memory cycle-stealing.

The main processor interprets and executes a standard minicomputer instruction set. The main processor controls activation of the display processor by means of special instructions.

The display CRT contains 2048 X 2048 (10) addressable points. Since it is a refresh CRT, screen images must be redrawn approximately 30 times a second to remain visible without "flickering". Hardware features for the IMLAC display include:

- Circle/arc generator
- Blinking
- Variable intensity
- Rotation/reflection
- Scaling
- Light pen

The dual disks are composed of one removable and one permanent cartridge. I/O is performed on a direct memory access (DMA) channel.

The Versatec printer/plotter can reproduce screen images, or it can be used to print alphanumeric characters. Dot matrices are generated via an electrostatic charge.

For further details on the IMLAC's physical charateristics, refer to the "PDS-4 System Manual".

2.2 Operating System

Several software utilities are provided to operate the PDS-4 system. Those utilities which directly support the described functions are summarized in the respective functional chapters. The central operating system module, termed the MONITOR, will be briefly discussed here.

As the system is initially powered up, a copy of the MONITOR is automatically loaded into memory and executed. When the MONITOR is in control, the system parameters are displayed along with the word "command". The command which the user types in will specify the name of a file to be executed. The MONITOR reads the user input, loads the appropriate file, and passes execution to that program. The loaded file overlays the MONITOR in memory. Upon completion of its task, or by user command, the executing file loads in a new copy of the MONITOR from the disk, and restarts its execution.

To support disk I/O, a set of tables and I/O routines remain permanently in core. This information is reinitialized whenever the system is powered on.

123

For further information on the IMLAC operating system, refer to the "Disk Operating System User's Manual".

2.3 File System

IMLAC files can reside on either the lower permanent disk (disk-$\emptyset$), or on the upper removable disk (disk-1). Different removable disks can be loaded for access to different files. Each file has a user area associated with it. These areas are logical in nature, and do not represent any particular physical location on the disk. There is no restriction placed on the amount of disk space allocated to an individual user area.

IMLAC files have a permission value associated with them. This value specifies which user area can access and modify the file. A full description is provided in the "Disk Operating System User's Guide".

Each file also has a two letter extension. This extension is used by several utility programs to indicate in which format the file is written, i.e. Fortran, object, data, etc. Files with a ".SV" extension belong to the operating system and should not be modified or deleted. Similarly, all files in user area 1,1 should not be altered.

2.4 Operating Procedures

When power is first applied to the IMLAC system, the user should make sure that the proper removable disk is loaded. This assures the user that files to be referenced are available. If the user does not intend to reference any

124

existing files, he should load a general purpose cartridge,
or disk, as a precaution against accidental "clobbering" of
a removable disk.

After the disk "ready" light comes on, the user can
initialize the operating system. This is accomplished using
the control console switches. First, the address switches
should be set to value 4Ø (octal); all switches except #1Ø
should be down. Next, press the stop button followed by the
start button. This will load in a copy of the MONITOR, and
the system parameters should be displayed on the screen.

The operator must now specify which user area and which
disk he wants to define as his primary source for file access.
This is critical since the operating system controls file
access based on which user area is specified. Any files
created are automatically defined as belonging to this parti-
cular user area and disk.

The user area is specified using the LOGIN program.
The user types in:

LO/n,m:p   n,m = user area (example: 2,3)
                          p   = primary disk (:Ø, :1, or
                                :Ø:1)

If the primary disk is the lower disk (:Ø), the user can not
access files on the removable cartridge. If the upper disk
is specified (:1), both upper and lower disks are attached for
access. When the user specifies both disks (:Ø:1), the first
disk is considered as primary, but files on both disks can be
used. All files created are placed on the primary disk, and
files referenced are searched for first in the user area on

the primary disk, then on the secondary disk.

When the system is initialized, the user is automatically logged into user area 1,1:∅. Although normal operations can be completed from this area, it is strongly recommended that the user log into his assigned user area. Similarly, if the system is already initialized, the new user should log into the appropriate area. This prevents the possibility of mistakes that wipe out another user's file, and ensures that any created files will belong to the correct user.

Upon completion of the desired tasks, the user may want to perform one or more of the following suggested procedures:

1. The user should verify that all files to be kept are located in the correct area of the removable disk (:1). Since the lower disk is utilized by everyone, it should be used only as a scratch pad to contain temporary or test files. THERE IS NO GUARANTEE TO ANY USER THAT A FILE ON THE LOWER DISK WILL REMAIN INTACT FOR ANY LENGTH OF TIME! The user should move all files to be retained to the upper disk.

2. The user should eliminate unnecessary or temporary files from his user area. This provides more space on the disk for other users, and prevents saturation of available space. This is especially important on removable cartridges shared with more than one user.

3. In order to safeguard user files on a removable disk, the user should remove the disk and load a general purpose cartridge for the next user. This practice prevents the general user from accessing another's file area.

126

4. The final recommendation involves shutting down the IMLAC system. No special commands are required; simply turn the power off. Care should be taken to stop the disk prior to turning the power off at the wall switch. Damage to the removable cartridge could result if the drive powers down without first being stopped.

3.1 Function Overview

Communication with a host machine is essential to provide the extended resources necessary for complex interactive graphics. This communication link allows the IMLAC to be used as an intelligent terminal. It is intelligent in the sense that the local computing power can perform pre- and postprocessing on transmitted information. This includes data formatting, macro commands, local editing, plus several additional features.

Although the actual data is sent and received through a hardware interface, it is necessary to have local executing software to interpret the input, run the display, and format the output. The IMLAC provides different programs (described in the next section) which perform these functions. The user simply calls one of these programs into execution, and proceeds to connect the communication link via the telephone modem. When the host system sends information through the link, the IMLAC program reads the characters, displays them, and waits for user input from the keyboard. This input is interpreted and sent to the host when applicable. Standard TTY communication is achieved with the benefit of a local program permitting improved performance.

Communication is not limited to any single host. The interface permits the user to specify such parameters as baud rate, data format, and type of communication. This allows the

IMLAC to connect with any computer that can be linked through the telephone modem. However, it is the user's responsibility to be familiar with the host system's communication protocol for intelligent conversation.

3.2 Available Software

3.2.1 PSA

PSA is a program which allows the IMLAC user to specify communication parameters. Baud rate, data format, and terminal type are programmable to allow communication with several different host machines. PSA executes in a question-answer format. Table 1 shows an example of the PSA values for CDC communication. Refer to the IMLAC manual "Disk Operating System User's Guide" for specific operating procedures.

TABLE 1

PSA Options for Cyber Communications

Option	Value
Terminal Type	A (TTY)
Send Baud Rate	G (12ØØ)
Receive Baud Rate	G (12ØØ)
Data Format	D (7/1Ø)
Data Level	A (8)

3.2.2 LINDI

LINDI is the main program for terminal communication. When executed, LINDI interprets several local commands. These commands modify the transmitting format, signify file transfers, and sets terminal parameters. Examples may be found in the "Disk Operating System User's Guide". One useful feature of this program is the ability to hold a communication link open. When the user is connected to a host machine, he can terminate the LINDI program (keyboard combination REP and Q keys), perform some IMLAC local function, recall LINDI into execution, and still have the communication link with the host.

Prior to loading the LINDI program, the user should check to ensure that the correct transmitting parameters are set. This is accomplished by running the PSA program, and specifying the desired parameters.

3.2.3 LIND2

LIND2 was created to provide terminal users with more capability and flexibility when using the IMLAC with a host machine. To use this program, simply type LIND2 while in the MONITOR command mode. Several features have been added to the IMLAC version of LINDI, and will be discussed below.

3.2.3.1 PSA Default Values

When LIND2 begins execution, the following question will appear on the screen:

OK TO USE DEFAULT PSA VALUES ? Y/N

If the user types Y in response, the following values are assigned to the interface: 12ØØ baud receive and send, 7/1Ø

130

format, and 8 level data. These values are based on a normal CDC TTY interface. If an N is responded, the values are not changed, and whatever was last specified via a PSA run or LIND2 execution is used. Thus, if some value other than default is to be specified, first run PSA and set the appropriate values, then run LIND2 and respond N.

3.2.3.2 Parameter Display

LIND2 displays several parameter values at the top of the screen when executing. The parameter list is illustrated in Table 2. To erase this display and allow the entire screen for terminal use, strike the FORM key. Successive strikes of this key will toggle the display. To erase the screen below the parameter display, strike the TAB key. This also erases the entire screen when the parameters are not being shown.

TABLE 2

LIND2 Parameter List

A) TTY Port

B) No Parity

C) Full Duplex

D) Echo

E) ASCII

F) Display On

G) Output File: None. $\emptyset$, $\emptyset$; $\emptyset$

H) Input File: None. $\emptyset$, $\emptyset$; $\emptyset$

3.2.3.3 Changing Parameters

Each parameter in the display has a letter associated with it. To modify any of these parameters, type:

#CH,n   or   #CHANGE,n   where n a letter A-H

The # specifies that it is a LIND2 command and no data is sent to the host until after a CR is struck. The chosen parameter will be toggled to a new value. For parameters G and H, an additional parameter must be specified in the CHANGE command following the letter and a coma. Some examples of this command are:

    #CH,A   - toggle the TTY/TKA mode
    #CH,G,FILA.DA 1,1:1   - change the input file name

3.2.3.4 Duplex Mode

The duplex mode can be toggled between full and half with a #CH,D command. When in full duplex, each character is sent to the host as it is typed (with the exception of LIND2 commands). To erase the previous character sent, strike the DEL key. This sends a CNTRL-H character to the host. Either a CR or XMIT can be used to specify a command terminator.

In half duplex, characters are sent only after a XMIT or PAGE XMIT key is struck. XMIT transfers all characters from the beginning of the current line to the cursor position. Thus, an entire line can be typed and edited before sending, or previous lines typed can be resent. PAGE XMIT transfers all characters from the home position to the cursor including all embedded CRs. If the parameter values are being displayed,

132

the home position is the first line after their display.
Using this capability, several lines can be edited and
sent at one time. Care should be taken when using half
duplex to ensure only the desired data is transferred.
Any extraneous information at the beginning of a line if
XMIT is used, or beginning of the screen if PAGE XMIT is,
will be sent to the host. Thus, if COMMAND-EDITOR is sent
with XMIT, the entire line is sent and will be misinterpreted
by the host. Examples of both modes are:

FULL DUPLEX:

```
    COMMAND-EDITOR(CR)          - types normal input
    CREO(DEL)ATE                - deletes error
```

HALF DUPLEX:

```
    COMMAND-(CR)               - (CR) starts new line so
    EDITOR(XMIT)                 that XMIT only sends the
      1ØØ=line1                  word EDITOR
      2ØØ=line2
      3ØØ=line3(PAGE XMIT)     - sending several lines
```

3.2.3.5 Sending Files from IMLAC to Host

To send an existing IMLAC file to the host machine, the
following command should be entered:

```
    #SEND,dsfn,lfn    -  dsfn is IMLAC file name
                         lfn is host file name
```

This command effectively sends a "COPYSBF,,lfn" to the host.
The file will be displayed on the screen as it is transferred,
and when it is finished, a %EOF is automatically sent telling
the host it is finished. The name of the IMLAC file sent will
be displayed in the parameter information. The format for the
IMLAC file name is:

133

```
fn.xx a,b:c;d    - where fn  = file name
                         xx  = two character extension
                         a,b = user area (such as 1,1)
                         c   = disk (∅ or 1)
                         d   = protection code
```

Only the file name and extension are required. The user area,
disk, and protection will default to whatever the user is
currently logged on. Care should be taken when attempting
to send files from some area other than that currently logged
on. If the protection on the file does not allow the user to
access it, the file cannot be sent. Some examples are:

```
#SEND,FILA.DA,LFN1        - sends FILA.DA in current user
                            area to host file LFN1
#SEND,FILZ.F4 3,2:1,LFN - sends FILZ.F4 from area 3,2 on
                            Disk-1 to host file LFN
```

3.2.3.6 Getting IMLAC Files from Host

To get a host file into an IMLAC file, the following
command should be entered:

```
#GET,lfn,dsfn    - where lfn  = host file name
                         dsfn = IMLAC file name
```

The format for the IMLAC file name is the same as specified
in the SEND section. This command will issue a "COPYSBF,lfn,
OUTPUT" to the host machine. The file will be displayed as
it is sent. When the transfer is complete (signified by ".."
in editor, or "COMMAND" in command mode), it is the user's
responsibility to strike LF which informs LIND2 that the
transfer is complete. No keyboard input is recognized until
the LF is struck. If the IMLAC file being created already
exists in the specified area, the following question appears
on the screen:

FILE ALREADY EXISTS, OK TO OVERWRITE ? Y/N

134

If Y is answered, the existing file is deleted before the transfer begins. If N is responded, the transfer is aborted. As in the SEND command, care should be taken in trying to get an IMLAC file in a different user area. Examples of this command are:

```
#GET LFN1,FILA.F4          - sends LFN1 to FILA.F4 in
                             current user area
    ..file..
COMMAND-(LF)               - when file finished LF sent

#GET LFN,FIL.DA 3,2:1;400  - sends LFN to FIL.DA on 3,2
                             Disk-1
    ..file..
    ..(LF)                 - when file finished LF sent
```

3.2.3.7 LIND2 Command Summary

Commands:

```
#CH,n             - changes parameter value

#SEND,dsfn,lfn    - transfers IMLAC to host file

#GET,lfn,dsfn     - transfers host to IMLAC file
```

Special Key Characters:

```
LF          - terminates GET

CR          - terminator in full duplex

XMIT        - transmits line in half duplex

PAGE XMIT   - transmits page in half duplex

FORM        - toggles display

TAB         - erases screen

HOME        - returns cursor to home

DEL         - deletes previous character

FNK 0       - increases screen character size

FNK 2       - decreases screen character size
```

135

3.2.4 STR14

STR14 simulates a Tektronix 4014 terminal. This permits interfacing to a host program written to drive a 4014 graphics screen. Individual commands can be found in the "STR14 User's Guide".

3.2.5 TIS4 and Others

TIS4 provides similar LINDI capabilities for TTY communication. However, it deals primarily with local and host graphics, and will be discussed in Chapters 6 and 7.

Other programs offer TTY communication as a means to initiate host interaction. Since these programs do not deal primarily with the communication function, they will be presented in the chapters dealing with their respective function. For strictly performing TTY functions, the user should employ LINDI, LIND2, or STR14.

4.1 Function Overview

To fully utilize the local computing power of the PDS-4, IMLAC has provided several software utilities for creating, editing, and compiling source programs. The primary storage device for these generated programs is the disk file. Source files can be created, loaded into memory and modified, compiled or assembled, and restored onto the disk.

File creation and editing are treated as a single function by IMLAC. When a user specifies an existing file to be edited, the file is loaded into memory and made available for user modification. When a non-existing file is requested for editing, the system presents a blank file on which the user can create the desired information. Information is entered via the keyboard and interpreted and stored as ASCII characters. If the user attempts to edit an existing file not in ASCII format, the screen will display unintelligible symbols (garbage). Once edited, the loaded file can be rewritten over the original file on the disk, saved onto a new file which does not destroy the original, or discarded completely. A means is provided which MAY salvage files discarded in error.

Once the ASCII file is stored in the desired format, it is ready to be accessed by the user. This may include being read as a data file by an executing program, compiled as Fortran into IMLAC assembly code, or assembled into object format. These aspects will be discussed under the "Program Execution" function.

4.2 DFED

The Disk Fast Editor program (DFED) is the main module for creating and editing ASCII files. DFED loads the specified file into memory and displays as many lines as possible on the screen. The user can scroll the lines up or down by moving the cursor with the keyboard arrows to view the entire file. Several editing features are provided by DFED, and outlined in the "Programmable Editor (DFED)" manual. Although the key sequence for using these features may appear awkward, it is strongly recommended that they be learned and utilized, for the capabilities offered provide a powerful tool for quickly generating and modifying files.

5.1   Funtion Overview

IMLAC provides a Fortran compiler, assembler, and binder to produce executable object code. This allows the user to write local Fortran source programs and execute them without interfacing to the host machine.

Several limitations prevent the use of the IMLAC compiler in the same general sense as the Cyber Fortran compiler. These limitations will be mentioned for the awareness of the new user.

One major problem is the lack of complete ANSI standard conventions. This eliminates the ability to execute the same Fortran program on both the IMLAC and the host machines. Non-ANSI standard differences are specified in the "Fortran User's Manual". Some examples are data statements, external statements, and complex numbers. These differences should be taken into consideration when creating a local Fortran file.

Another limitation is the awkward subroutine handling. Each subroutine must be compiled separately in a different file, then referenced when the program is run. The problem is heightened by the size limitation on any single program segment. If the Fortran compiles into assembly code longer than 4000 (8) locations, errors result in the assembly phase for the program. This forces the programmer to break a large Fortran program into separate subroutines before compilation.

Graphic capability is another area lacking sophistication.

The local Fortran does support basic graphic functions as described in Chapter 14 of the "Fortran User's Guide", but these functions are limited and do not take advantage of the versatile graphics hardware available.

Because of these compiler limitations, the user needs to determine how to best utilize the available capabilities. One suggestion is to use the local Fortran as a post-processor of host data. Host data can be locally interpreted, summarized, and formatted for user applications. Another possible Fortran use would be to generate input data for host programs. In both examples, the Fortran should be developed specifically for the IMLAC system. The IMLAC "Fortran User's Guide" should be closely referenced for specific capabilities.

The compilation of a program generates an assembly language file. This file must now be assembled to generate object code. IMLAC provides an assembler to perform this function. The assembler recognizes IMLAC mnemonics and special assembler macros. It translates the program into machine code in one of two possible formats, object or relocatable.

Object code is completely self-contained and ready for execution. Relocatable code contains undefined references which must be satisfied prior to execution. The process of satisfying these references is termed linking or binding. Binding is performed by a separate utility program which interprets which files are to be used for satisfying external references, and completes the object code.

The final object code is the version of the program which

140

is actually executed by the IMLAC. When the user types in the name of a file (.OB extension is assumed), IMLAC searches the disk for the file, loads it into memory, and passes execution to it.

The normal sequence of the execution is file editing, compiling, assembling, and linking. The corresponding program formats are Fortran source, assembly language, relocatable, and object.

5.2 Available Software

5.2.1 COMPIL Subsystem

When a user generates a Fortran source file (specified by a .F4 extension), the compilation can be accomplished by the one word command COMPIL. COMPIL will perform the compilation and check for any errors reported by the Fortran compiler. If any errors are detected, they will be displayed for the user. If no errors are found, the subsystem produces an assembly language version of the program to be assembled. Refer to the "Fortran-IV Language Manual" for details of this subsystem.

5.2.2 FCOMP

FCOMP is the name of the Fortran compiler, and is called by the COMPIL subsystem. The user can execute FCOMP directly and perform his own error checks.

5.2.3 RUN Subsystem

RUN is a subsystem which uses the assembly language file from the compilation, assembles it, binds it with the Fortran

141

library, and executes the final object code. This command
usually follows the COMPIL subsystem. Specifics on the sub-
system can be found in Appendix A of the "Fortran-IV Language
Manual".

5.2.4 ASM

ASM is the IMLAC assembler. It is called by the RUN
subsystem, or can be executed directly by the user. Several
macros and options are recognized as presented in the "Disk
Assembler (ASM) User's Guide". If a program is to be written
directly in assembly code, the user should first study the
"PDS-4 System Reference Manual" to become familiar with the
operating concepts of the system.

5.2.5 BIND

BIND is the IMLAC linker for producing executable object
code. This utility is called by the RUN subsystem for bind-
ing Fortran routines to an assembled program. Any user writ-
ing assembly language code can use BIND directly to reference
existing routines in the assembly program. Details of opera-
tion can be found in the "Linking Editor (BIND) User's Guide".

5.2.6 DIDL

DIDL is a debugging utility which allows a user to find
errors in a program while executing it. DIDL provides the
ability to step through the program's execution, check variable
values, and directly query memory locations. This is an ex-
tremely useful tool for the sophisticated user. The "Disk
Interactive Symbolic Debugger (DIDL) User's Guide" presents

specific information on the program's use.

6.1 Function Overview

"Local graphics" is a term used to indicate the production of graphic displays without interfacing to a host computer. Specifically, a set of display commands must be created for the display processor to execute. This set of commands, called the display list, can be created in several ways. The most direct way to create a display list is to directly code it in assembly code. This method offers the full range of capability provided by the display hardware. However, it is impractical for all IMLAC users to learn the details of the assembly language. This method is recommended to the experienced assembly programmer for its flexibility and efficiency.

Another means for generating displays is to execute a Fortran program containing display commands. The display lists developed by this method are not as efficient as if directly coded, but require less knowledge on the part of the user. The major drawback to this technique is the inability to use the more sophisticated hardware capabilities, such as circle generation, or the automatic reflection and rotation. The list of available Fortran routines is located in the "Fortran-IV User's Manual".

The final means for generating displays is IMLAC's provided utility programs. These utilities are placed in execution, and respond to user commands to draw lines, circles, and alphanumerics. In addition, subpictures can be defined

for rotation, scaling, and translation. The final display image can be saved to a disk file for future reference.

6.2 Available Software

6.2.1 TIS4

TIS4 is the most versatile program available for creating complex displays. It can be used both interactively with a host, or in a local mode. The host mode is discussed in the host graphics section. When using it locally, TIS4 accepts commands from the keyboard to create figures, edit subpictures, and load or store displays. The light pen capability is used to select subpictures for editing. The complete command set is found in the "Terminal Interface System (TIS) User's Guide".

6.2.2 SYMBL4

SYMBL4 is used to define and store entire sets of symbols. A grid is produced on the screen to allow the user to generate symbols 16 times their normal size. These symbols can be saved in a local inventory or saved on a disk file. The format for saving the information to disk is specified by the user as either source or object. The source code is in display assembly language, and can be directly inserted into an assembly program or Fortran program using the embedded assembly code feature. Instructions for SYMBL4 usage can be found in the "SYMBOLFORM Character Generating Program User's Guide".

7.1 Function Overview

A major advantage of computer graphics is the ability to interact with an executing program through the display. "Host graphics" refers to this interaction between the IMLAC display screen and an executing program on a host computer. The graphics system on the IMLAC must be able to accept and interpret commands from the host, perform the display functions, and transmit information from the user back to the host.

IMLAC provides this capability in a fashion similar to its TTY programs. A utility program is executed which can receive and transmit data to a host through the modem. The user must log into the host system, and place a host program into execution. This program generates and sends graphic commands to the IMLAC. The local utility interprets the commands, and performs the desired functions. If the command requests user interaction, the local utility will accept the user response and transmit it back to the host.

7.2 Available Software

7.2.1 TIS4/GCS

TIS4 was briefly described in the local graphic section. It has an interactive capability which was specifically designed to interpret commands from the Graphics Compatability System (GCS) library. To use TIS4, the user creates a Fortran program on the host machine which contains calls to the GCS library routines. The user then executes the TIS4 utility

which has a TTY function, and logs into the host. Before executing the host program, the user must attach the appropriate library which contains the GCS driver for the IMLAC. When the host program is executed, this driver will transmit the graphic commands to TIS4 to perform. Specific details for this function are available in the "Graphics Compatability System (GCS) Programmer's Manual" and the "TIS User's Guide".

7.2.2 GTS/GRAPHELP

GTS operates similarly to TIS4. GTS is executed on the IMLAC, and it allows TTY communication for logging onto the host. The graphic commands from the host are generated using the GRAPHELP library. These calls are different from those available in GCS; three dimensional commands are not available, and different formats exist. These library routines were written for a 16-bit minicomputer, and are not available for CDC host computer operation.

8.1 Function Overview

IMLAC provides several utilities to help the user manage
the disk file system described in Chapter 2. These utilities
perform such tasks as file modification, directory querying,
and file dumping. All utilities listed are presented in detail
in the "Disk Operating System User's Manual".

8.2 Available Software

DELETE - removes a file from the disk.

GET   - retrieves a file to the current user area from another
           area or disk. This utility is used to save files to
           a removable disk.

MN/NM - translates files from ASCII to binary or vice-versa.

PK/KP - pack and unpack information on files.

DIREKT - displays files contained in a specific user area.

DPRINT - prints the directory to a printer.

PRINT - prints an ASCII file to the printer.

RENAME - changes the name of an existing file.

FLUSH - deletes all files not belonging to user area 1,1.
            This utility should be USED WITH CARE.

Several other programs are available for copying entire
disks, error checking, and specifying system parameters.
These functions should be used with care because of their
ability to "clobber" file information. Reference the oper-
ating manual for specific instructions.

The following programs are not involved directly with any of the described functions, and are mentioned here for user reference.

9.1 JOD

JOD is the IMLAC text editor program. It relies mainly on the DFED commands for entering and editing text. Once entered, the information can be automatically justified, paged, and contented. The "JOD Text Editing System User's Guide" contains specific details for using the system.

9.2 SIZE

SIZE is a locally created utility to allow the user to select the screen print size for the MONITOR. The format for executing this utility is:

SIZE/n   - where n is a scale from 1-7

This parameter will only affect the screen size when the MONITOR is in execution.

9.3 MSG

MSG was locally developed to permit a one line message to be displayed with the system parameters of the MONITOR. The format for this function is:

MSG/message to be displayed...

The MONITOR will display the same message until changed with another MSG command.

Vita

Dennis L. Schweitzer was born on 11 September 1952 in Canton, Ohio. After graduating from Lehman High School in 1970, he continued his education at the United States Air Force Academy, Colorado Springs, Colorado. On 5 June 1974, he was graduated with the degree of Bachelor of Science and was commissioned a Lieutenant in the United States Air Force. Following graduation, he spent three years as a computer systems analyst at Headquarters, Strategic Air Command, Offutt AFB, Omaha, Nebraska. He was assigned to the Air Force Institute of Technology in August, 1977.

Permanent address: 5612 Brentwood Ave.
North Canton, Ohio 44720

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/79-3	**2. GOVT ACCESSION NO.**	**3. RECIPIENT'S CATALOG NUMBER**
4. TITLE (and Subtitle) EVALUATION OF AN EXISTING COMPUTER SYSTEM USING STRUCTURED ANALYSIS TECHNIQUES		**5. TYPE OF REPORT & PERIOD COVERED** MS THESIS
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dennis L. Schweitzer Captain USAF		**8. CONTRACT OR GRANT NUMBER(s)**
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		**10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS**
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Flight Dynamics Laboratory (FBR) Wright-Patterson AFB, Ohio 45433		**12. REPORT DATE** Mar 1979
		13. NUMBER OF PAGES 158
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		**15. SECURITY CLASS. (of this report)** Unclas
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE AFR 190-17.

JOSEPH P. HIPPS, Major, USAF   16 MAY 1979
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Software
Structured Analysis
Software Requirements
Requirements Analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis is an evaluation of an existing computer graphics system using structured analysis techniques. The system evaluated is the IMLAC PDS-4 minicomputer at the Air Force Flight Dynamics Laboratory, Wright-Patterson AFB.

The technique used in the evaluation is performed in three steps. First, the existing system is modeled into its

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73

Block 20.

logical equivalent using bubble charts. This model shows the flow of data through the system, and the activities performed on that data. Second, a similar model is produced describing the required system as defined by the users. This model is a representation of the users' desires and requirements, and is similar to a model which might be produced during the requirements definition phase of a computer's life cycle. The third step is to compare the two models to evaluate the existing system's effectiveness, and reveal areas requiring modification for full utilization of the system.

Following the evaluation, several recommendations and changes are proposed to increase the system's capability to meet the user requirements.